Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 05-001

An Efficient Parallel Finite-Element-Based Domain Decomposition Iterative Technique With Polynomial Preconditioning

Yu Liang, Ramdev Kanapady, and Kumar Tamma

January 18, 2005

Report Documentation Page				*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE **18 JAN 2005**	2. REPORT TYPE	3. DATES COVERED **-**

4. TITLE AND SUBTITLE **An Efficient Parallel Finite-Element-Based Domain Decomposition Iterative Technique With Polynomial Preconditioning**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Army Research Laboratory,2800 Powder Mill Road,Adelphi,MD,20783-1197**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
The original document contains color images.

14. ABSTRACT
see report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**		**29**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

An Efficient Parallel Finite-Element-Based Domain Decomposition Iterative Technique With Polynomial Preconditioning

Yu Liang [*], Ramdev Kanapady [†] and Kumar K. Tamma [‡]

Abstract

An efficient parallel finite element-based domain decomposition iterative technique with polynomial preconditioning with particular attention to the GMRES solver is presented. Unlike the standard row-oriented partitioning of a matrix, finite element based domain decomposition with polynomial preconditioning circumvents the assembly of matrix, reordering of matrix, redundant computations associated with the interface elements, numerical problems associated with local preconditioner, and costly global preconditioner construction. A dramatic reduction in parallel overhead both in terms of computation and communication results in a highly scalable solver. The parallel performance results for large-scale static and dynamic problems on the IBM SP2 and the SGI Origin are presented.

Keywords: GMRES, domain decomposition, distributed format, polynomial preconditioner, finite element, linear equations.

1 Introduction

Parallel iterative solvers are better alternative than the parallel direct solvers for large-scale finite element computations [1, 2]. Numerically scalable domain decomposition based solvers such as *finite element tearing and interconnecting* (FETI) method [3,4] are better suited for highly parallel finite element computations. This however is mainly restricted to symmetrical system arising from finite element formulations for solid and structural mechanics based applications. The iterative solvers such as GMRES method [5] are an effective method for iterative solution of specific classes of un-symmetric and symmetric linear systems arising from finite element formulations [1]. The rate of convergence of the iterative algorithms is governed by the condition number of the linear systems, which increases dramatically for finite element matrices as the mesh size decreases. As a result, an efficient preconditioner is required for the solution of the linear systems. Finite element-edge based domain decomposition (DD) provides an intuitive while powerful framework for the parallel formulation of finite element method (FEM) [6]. Within this DD framework providing an efficient preconditioner for an iterative solver is a challenging task. The Sparse Approximate Inverse (SPAI) [7] and diagonal preconditioners are commonly used preconditioner for parallel formulations. However, SPAI is not suitable for finite element edge based domain decomposition strategies [8] and diagonal preconditioners are not effective enough to reduce the number of iterations for large-scale complex problem. Incomplete LU factorization (ILU(k)) [7] based preconditioners are effective for single processors computational framework. However, for parallel framework, its use is mainly limited as it incurs high parallel communication overhead similar to that of direct solvers, and in particular for the finite element edge-based DD framework. The polynomial preconditioners [9] are simple, yet efficient and effective methods for efficient parallel iterative solvers. Polynomial preconditioning methods has been investigated in literature [5, 7, 9–16]. In parallel framework, polynomial preconditioning methods are mainly associated with row partitioning of the matrix of the linear system.

In this paper, an efficient parallel finite element-based domain decomposition GMRES solver in conjunction with polynomial preconditioning is presented. The distinguishing features of the proposed approach is that it circumvents:

[*]Postdoc Associate, yliang@cs.umn.edu, Army High Performance Computing Research Center.

[†]Research Scientist, ramdev@me.umn.edu, Department of Mechanical Engineering, and the Army High Performance Computing Research Center.

[‡]Correspondence to: Professor and Technical Director, ktamma@tc.umn.edu, Department of Mechanical Engineering and the Army High Performance Computing Research Center, University of Minnesota, Minneapolis, MN 55455. e-mail:ktamma@tc.umn.edu

i) the assembly of the finite element coefficient matrix and preconditioner associated with subdomain interface, ii) reordering of a matrix to gain parallel performance, iii) redundant computations associated with the interface elements, and iv) numerical problems associated with local ILU(k) such as singularity preconditioner. This results in a dramatic reduction in the parallel overhead both in terms of computation and communication and results in a highly scalable solver. The flexible GMRES with Neumann series and generalized least-squares polynomial preconditioner [15] are employed here for large-scale static 2nd order elasticity and elastodynamics problems on an IBM SP2 and SGI Origin to illustrate the effectiveness and parallel performance of the proposed approaches.

The remainder of this paper is organized as follows. In Section 2 the sequential preconditioning techniques with focus on polynomial preconditioning methods for their subsequent parallel formulation is presented. Section 3 discusses the parallel implementation of a finite element based domain decomposition flexible GMRES solver followed by the discussions on parallel implementation of a flexible GMRES solver for row-based domain decomposition techniques in Section 4. In Section 5 the communication/computation complexity of the finite element-based and node-based flexible GMRES is briefly discussed to highlight the advantages of the finite element-based domain decomposition based solver. In Section 6, the convergence and parallel performance for static and dynamic problems employing the present parallel polynomial preconditioning flexible GMRES is provided followed by conclusions in Section 7.

2 Preconditioned Iterative Solvers

In this section, the commonly used preconditioning techniques with particular attention to the polynomial preconditioning methods are presented. The polynomial preconditioning techniques such as the Neumann series and the generalized least-squares (GLS) methods are discussed. The construction and factors affecting the numerical stability of these polynomial preconditioners are also discussed. An important pre-processing technique such as diagonal scaling to effectively apply the polynomial preconditioning is also discussed which is subsequently employed in the parallel implementation of both element- and node-based partitioning based strategies.

2.1 Preconditioning methods

Implicit finite element computations in several scientific and engineering problems such as linear/nonlinear, static or dynamic problems requires the resulting solution of a system of linear equations such as

$$\mathbf{K}\mathbf{u} = \mathbf{f} \tag{1}$$

where $\mathbf{K} \in \Re^{N\times N}$ is a sparse coefficient matrix, $\mathbf{u} \in \Re^{N}$ is the vector of unknowns and $\mathbf{f} \in \Re^{N}$ is the load vector and N is the number of equations. In the case of large-scale problems, domain decomposition (DD) provides an intuitive yet powerful framework for the parallel formulation of the finite element method (FEM). If Ω is the original problem domain which is partitioned into $\mathcal{P}$ sub-domains, then Eq. 1 can be written as

$$\left\{\bigcup_{s=1}^{\mathcal{P}} \mathbf{K}^{(s)}\right\}\left\{\bigcup_{s=1}^{\mathcal{P}} \mathbf{u}^{(s)}\right\} = \left\{\bigcup_{s=1}^{\mathcal{P}} \mathbf{f}^{(s)}\right\} \tag{2}$$

where $\mathcal{P}$ is the number of processors, $\bigcup_{i=1}^{\mathcal{P}}$ denotes the “assembly” operation, $\mathbf{K}^{s}$ is the coefficient matrix, $\mathbf{K}^{s}$ and $\mathbf{u}^{s}$ is the solution vector, and $\mathbf{f}^{s}$ is the right-hand side vector associated with the each partition s. Here partition could be finite-element-node based or finite-element-edge based partition. The former leads to row-partition of the associated matrix. This forms the basis for most of the parallel implementations of the iterative solvers [7, 17, 18]. In this paper we assume the finite element-edge based parition and each sub-domain is mapped into each processor involved in the computation.

The rate of convergence of the iterative algorithms are governed by the condition number of the resulting linear systems, which increases dramatically for finite element matrices as the mesh size decreases. As a result, an efficient preconditioner is required for the solution of the linear systems in Eq. 1. If $\mathbf{C}$ is a preconditioner, then Eq. 1 can be transformed into

$$\begin{aligned} \mathbf{C}_L\mathbf{K}\mathbf{u} &= \mathbf{C}_L\mathbf{f}, & (3)\\ \mathbf{K}\mathbf{C}_R\mathbf{v} &= \mathbf{f}, \mathbf{u} = \mathbf{C}_R\mathbf{v},\ \textit{or} & (4)\\ \mathbf{C}_L\mathbf{K}\mathbf{C}_R\mathbf{v} &= \mathbf{C}_L\mathbf{f}, \mathbf{u} = \mathbf{C}_R\mathbf{v} & (5) \end{aligned}$$

where $\mathbf{C}_L$, $\mathbf{C}_R$ are the preconditioning matrices such that $\mathbf{C}_L\mathbf{K}$, $\mathbf{K}\mathbf{C}_R$ and $\mathbf{C}_L\mathbf{K}\mathbf{C}_R$ are approximately equal to the identity matrix $\mathbf{I}$. The Eqs. 3, 4 and 5 are called left-, right- and split-preconditioners [7] respectively. In this paper, our attention is restricted to left- and split-preconditioners. Serial implementation of the commonly employed preconditioners are incomplete LU factorization (ILU(k), where k is the level of fill-in). For high parallelizability, Sparse Approximate Inverse (SPAI) and diagonal preconditioners are commonly used. However, SPAI is not suitable for element-based domain decomposition strategies and diagonal preconditioners are not effective enough to reduce the number of iterations for large-scale complex problems.
Diagonal scaling preconditioner is generally employed in the split-preconditioning scheme so as to keep the symmetry of linear systems. It is easy to construct and incur little communication in its application. However, except for a few cases of diagonally dominant linear systems, diagonal scaling preconditioner cannot effectively reduce the number of iterations. Diagonal scaling operation incur no communication overhead and is available for element-based domain decomposition (EDD) method. In this paper, diagonal scaling preconditioner is utilized as an indispensable pre-processing tool for the polynomial preconditioner.

The interest in polynomial preconditioners [15] is motivated by the need for simple, yet efficient and effective methods for efficient parallel iterative solvers [7] for high performance computers (HPC) such as vector and parallel processors. In contrast to ILU, SPAI, etc. polynomial preconditioners are constructed only based on the the matrix spectrum (i.e., eigvalues of the coefficient matrix), which is denoted as $\sigma(\mathbf{K})$. Even though $\sigma(\mathbf{K})$ is generally difficult to compute, Θ, an approximate estimation to it can be easily obtained. When $\mathbf{K}$ is a symmetric matrix, then $\Theta \subset \Re$; otherwise $\Theta \subset \mathcal{C}$. The accuracy of Θ determines the rate of convergence of the preconditioned systems. Since polynomial preconditioning will always keep the symmetry of linear systems [15], in this paper we employ left-preconditioning, Eq. 3, to transform the original linear systems, Eq. 1, to yield

$$P_m(\mathbf{K})\mathbf{K}u = P_m(\mathbf{K})\mathbf{f} \tag{6}$$

where $P_m(\mathbf{K})$ is a polynomial in $\mathbf{K}$ with degree $\leq m$. It has been shown in [15] that, if $P_m(\lambda)$ is constructed such that

$$\min_{P_m \in \wp_m[\Theta]} \|1 - \lambda P_m(\lambda)\| \quad (\lambda \in \Theta) \tag{7}$$

where $\wp_m[\Theta]$ is the set of m-degree polynomials which is valid over Θ, and $\|.\|$ represents a specific norm, namely, the uniform norm or quadratic norm, then a bound on the condition number [7] of the preconditioned systems, $\kappa(P_m(\mathbf{K})\mathbf{K})$, is minimized.

In this paper the coefficient matrix $\mathbf{A}$ is assumed to be symmetric. The difficulty in applying the polynomial preconditioner lies in the estimation of the spectrum of coefficient matrix. This can be easily achieved by a simple pre-processing technique that maps the spectrum of the coefficient into a predetermined interval. This is discussed in the next section.

2.1.1 Diagonal scaling

Scaling is an essential pre-preprocessing technique applied to Eq. 1 before the polynomial preconditioning stage. An appropriate scaling operation is essential for ready application of the polynomial preconditioner. Besides reducing the condition number of linear systems, it restricts the spectrum of the coefficient matrix $\sigma(\mathbf{A})$ into a pre-determined interval.

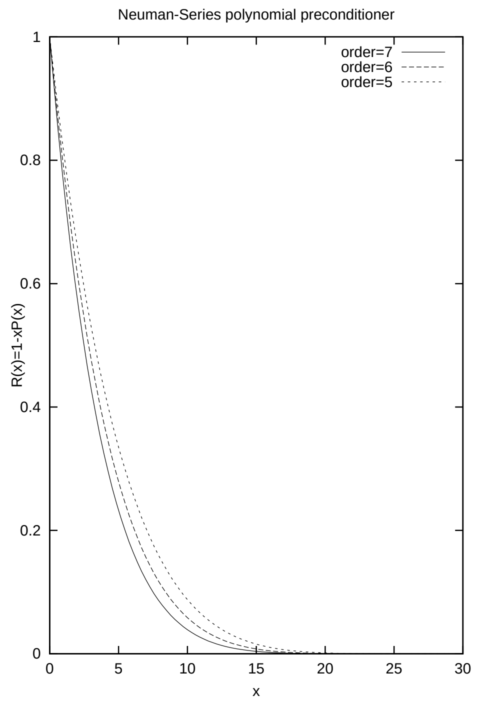


Figure 1: Residual polynomial of the Neumann-series preconditioner.

Theorem 1 *Let* $\mathbf{K} = [\mathbf{k}_1, \mathbf{k}_2, \cdots, \mathbf{k}_N]^T \in \Re^{N\times N}$ *be an irreducible matrix and assume that* $\lambda_{\max}$ *is the largest eigenvalue of K, then*

$$\lambda_{\max} \ \le \ \max_{i=1}^{N} \|\mathbf{k}_i\|_1, \ \ \mathbf{k}_i \in \Re^N \tag{8}$$

where $\|\mathbf{k}_i\|_1 = \sum_{j=1}^{N} \mid k_{i,j} \mid$ *is called the discrete* $\mathcal{L}_1$ *norm of vector* $\mathbf{k}_i$.

For the proofs of the above theorem, the reader is referred to the Gershgorin theorem [7]. Given a linear system, Eq. 1, consider $\mathbf{K} = [\mathbf{k}_1, \mathbf{k}_2, \cdots, \mathbf{k}_N]^T$, and

$$\mathbf{D} = \text{diag}(\frac{1}{\sqrt{d_1}}, \ldots, \frac{1}{\sqrt{d_N}}) \tag{9}$$

where

$$d_i = \|\mathbf{k}_i\|_1 = \sum_{j=1}^{N} \mid k_{i,j} \mid \tag{10}$$

Then, the Eq. 1 can be transformed into

$$\mathbf{Ax} = \mathbf{b} \tag{11}$$

where $\mathbf{A} = \mathbf{DKD}$, $\mathbf{b} = \mathbf{Df}$ and $\mathbf{x} = \mathbf{Du}$. According to Theorem 1, it follows that

$$\sigma(\mathbf{A}) \subset (0,1) \tag{12}$$

This enables the construction of the polynomial preconditioner for Eq.11 such that $\Theta = (0,1)$. In the following two sub-sections, the Neumann series and the generalized least-squares preconditioning polynomial are discussed.

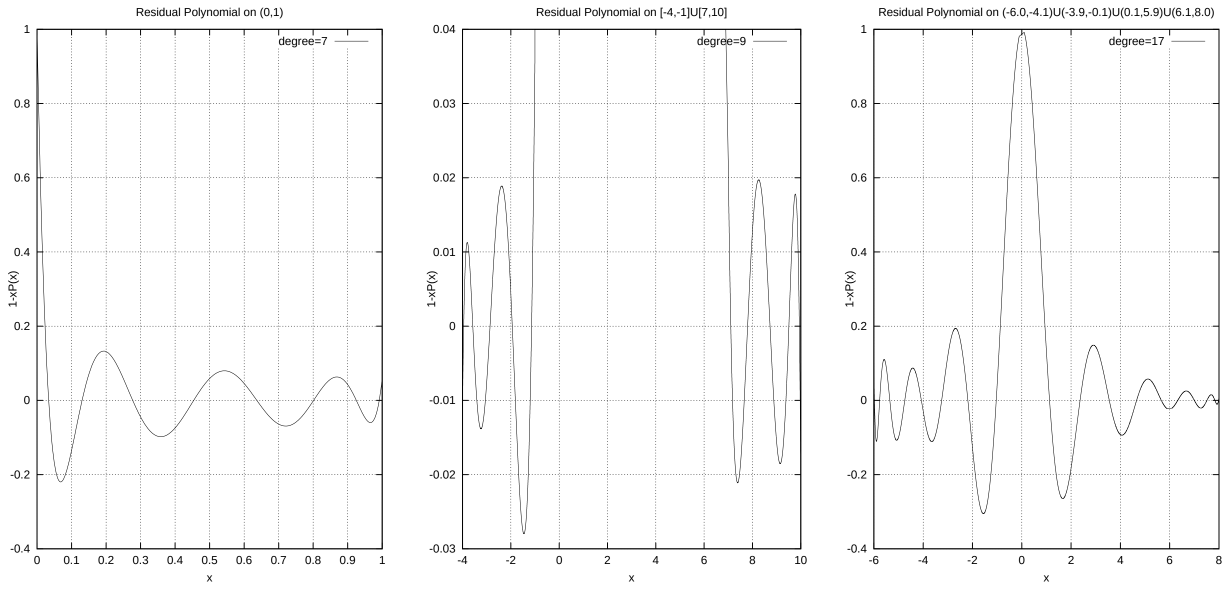


Figure 2: Residual polynomial $1-\lambda P_m(\lambda)$ for various Θ and polynomial degrees: a) $\Theta = (0.1, 2.5)$, b) $\Theta = (-4,-1) \cup (7,10)$, and c) $\Theta = (-6.0,-4.1) \cup (-3.9,-0.1) \cup (0.1,5.9) \cup (6.1,8,0)$.

2.1.2 Neumann series polynomial preconditioner

The Neumann polynomial preconditioner is the simplest polynomial preconditioner and it originates from the basic algebraic relation

$$\frac{1}{1-\lambda} = \sum_{i=0}^{\infty} \lambda^i \qquad \forall \mid \lambda \mid < 1,\ \lambda \in \Re \tag{13}$$

The following theorem may be used in the construction of a polynomial approximation to the inverse of the coefficient matrix $\mathbf{A}$.

Theorem 2 *Let* $\mathbf{G} \in \Re^{N \times N}$, *then* $\sum_{k=0}^{\infty} \mathbf{G}^k$ *converges if and only if* $\rho(\mathbf{G}) < 1$ *where* $\rho(\mathbf{G})$ *is the spectral radius of matrix* $\mathbf{G}$. *Then, it readily follows that*

$$\sum_{k=0}^{\infty} \mathbf{G}^k = (I - \mathbf{G})^{-1} \tag{14}$$

Let $\omega\mathbf{A} \equiv \mathbf{I} - (\mathbf{I} - \omega\mathbf{A})$ and $\mathbf{G} = (\mathbf{I} - \omega\mathbf{A})$ where $\omega \in \Re$ is the scaling factor. Then, from Theorem 2, the inverse of any coefficient matrix $\mathbf{A}$ is given by

$$\mathbf{A}^{-1} = \omega(\mathbf{I} - \mathbf{G})^{-1}. \tag{15}$$

In Eq.15 ω can be adjusted so that $\rho(\mathbf{G}) < 1$. Then it follows that $\mathbf{A}^{-1}$ may be approximated as follows:

$$\mathbf{A}^{-1} = \omega(\sum_{i=0}^{\infty} \mathbf{G}^i) \approx \omega(\mathbf{I} + \mathbf{G} + \cdots + \mathbf{G}^{m-1}) \tag{16}$$

Since the preconditioner matrix $P_m(\mathbf{A}) \approx \mathbf{A}^{-1}$, from Eq. 16 it follows that, the preconditioner $P_m(\mathbf{A})$ is given by

$$P_m(\mathbf{A}) = \omega(\mathbf{I} + \mathbf{G} + \mathbf{G}^2 + \cdots + \mathbf{G}^m) \tag{17}$$

In practical implementations, ω would be determined according to the characteristics of $\mathbf{A}$. For example, if $\mathbf{A}$ is symmetric positive definite then one can select $\Theta = (0, \hbar)$. Figure 1 shows that the residual polynomials $(1 - \lambda P_{m-1}(\lambda)), m = 5, 6, 7$ is approximately equal to zero in the interval $\Omega = (0, 30)$.

2.1.3 Generalized least-squares polynomial preconditioner

The generalized least-squares (GLS) polynomial preconditioning is preferable to other polynomial preconditioning methods (i.e., Neumann series, least-squares, Chebyshev etc.) due to its high application flexibility and low construction cost. For GLS polynomial preconditioning method, Θ is defined as a union of an arbitrary number of disjoint intervals of the spectrum, i.e.,

$$\begin{aligned} &\Theta = \bigcup_{k=1}^{N_I} (\ell_k, \hbar_k), 0 \notin \Theta \quad \text{and} \\ &\ell_1 < \hbar_1 \le \ell_2 < \hbar_2 \le \cdots \le \ell_{N_I} < \hbar_{N_I}, \end{aligned} \tag{18}$$

where N_I is the number of intervals in which the eigenvalue of the coefficient matrix $\mathbf{K}$ lies. Therefore, the GLS method can be a general method of solving symmetric linear systems including both symmetric indefinite and symmetric positive definite systems.
Once Θ is determined, the GLS polynomial preconditioner can be constructed such that

$$\min_{P_m \in \wp_m[\Theta]} \|1 - \lambda P_m(\lambda)\|_w, \;\; \lambda \in \Theta \tag{19}$$

where $\|.\|_w$ represents the weighted quadratic norm induced by the inner product $\langle f, g\rangle_w = \int_\Theta f(\lambda) g(\lambda) w(\lambda) d\lambda$ and $w(\lambda)$ is a non-negative weight function over the interval Θ. The key to the above least-squares problem, Eq. 19, is the construction of a sequence of orthogonal polynomials. Given a series of orthogonal polynomial sequence $\{\lambda\phi_i(\lambda)\}_{i=0}^m$ with respect to the inner-product $\langle , \rangle_w$ [15] as a basis, then the least-squares problem, Eq. 19, can be readily constructed via

$$P_m(\lambda) = \sum_{i=0}^{m} \mu_i \phi_i(\lambda), \tag{20}$$

where $\mu_i = \langle 1, \lambda\phi_i(\lambda)\rangle_w$. As a result, the polynomial preconditioning can be computed iteratively as

$$P_m(\mathbf{A})\mathbf{v} = \sum_{i=0}^{m} \mu_i \phi_i(\mathbf{A})\mathbf{v}. \tag{21}$$

In this work the orthogonal polynomial sequence $\{\lambda\phi_i(\lambda)\}_{i=0}^m$ is efficiently constructed using Chebyshev polynomials [15] embedded in the Stieltje procedure via the three-term recurrence relationship. The storage and time cost for this operation is negligible compared to ILU, SPAI, etc. For more detailed information about the sequential implementation of the polynomial preconditioner, the reader is referred to [15].
Figure 2 (a),(b) and (c) show the graphs of the residual $1 - \lambda P_m(\lambda)$ for the GLS polynomial preconditioning with respect to different Θ and m. It can be readily shown that, if $\Theta \approx \sigma(\mathbf{K})$, then $P_m(\mathbf{K})\mathbf{K} \approx \mathbf{I}$ leading to an efficient preconditioner. For the iterative solvers, the polynomial preconditioning step

$$\mathbf{z} = P_m(\mathbf{A})\mathbf{v} \tag{22}$$

reduces to a set of m matrix-vector product operations instead of incomplete factorization followed by backward and forward solve operations. In such a case, the only requirement that needs to be satisfied is the stability of the preconditioner, which is discussed in the following section.

2.2 Stability of polynomial filtering

Let ε be the machine precision (roundoff unit) and let

$$P_m(\mathbf{A})\mathbf{v} = \left(\sum_{i=0}^{m} \breve{a}_i \mathbf{A}^i\right)\mathbf{v} \tag{23}$$

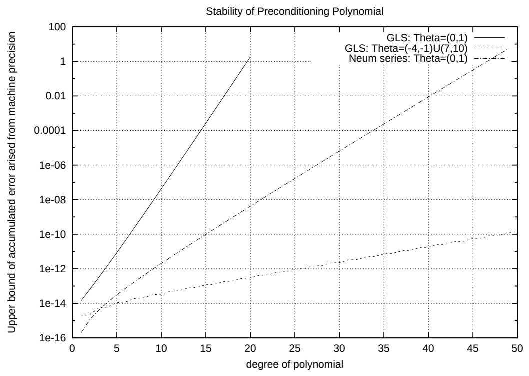


Figure 3: $\sum_{i=0}^{m} \mid \check{a}_i \mid$ for GLS preconditioning polynomial.

Let $\mathbf{z}_{fl} = P_m(\mathbf{A})\mathbf{v}$ satisfy the floating point arithmetic, and $\mathbf{z} = P_m(\mathbf{A})\mathbf{v}$ be the exact value. Then it follows from [16],

$$||\mathbf{z}_{fl} - \mathbf{z}||_2 \leq m\varepsilon \sum_{i=0}^{m} \mid \check{a}_i \mid . \tag{24}$$

The inequality Eq. 24 indicates that the stability of the preconditioning operation $P_m(\mathbf{A})\mathbf{v}$ mainly depends on m, the degree of the preconditioning polynomial P_m and the sum of the absolute values of the coefficients of the polynomial. Figure 3 shows the norm, Eq. 24, of various $P_m(\lambda)$ with increasing degree of polynomial m for $\Theta = (0,1)$ and $\Theta = (-4,-1) \cup (7,10)$. It can be seen that, for the GLS polynomial preconditioning, the accumulated error increases dramatically as the degree of polynomial increases. It is clear that for all practical purposes the degree of the polynomial should be restricted to less than 10.

2.3 Sequential flexible GMRES

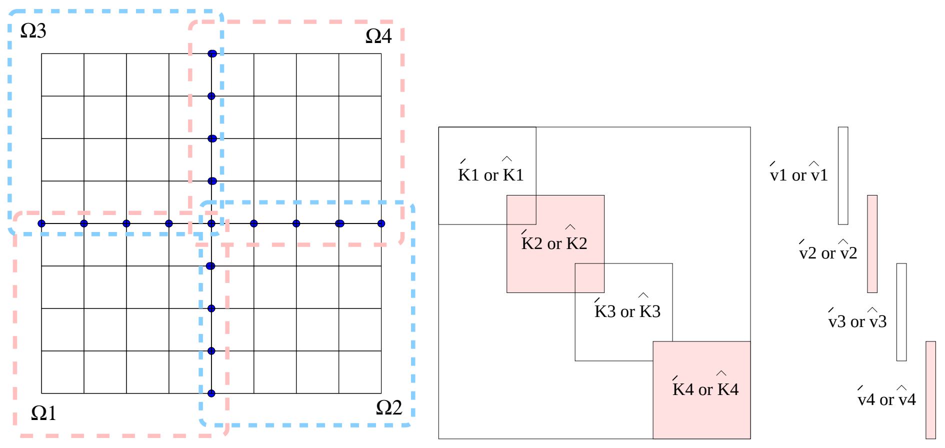
In this paper, the generalized minimal residual solver is chosen due to its general applicability to a wide variety of applications including problems arising in numerical formulations of structural mechanics applications. To formulate the parallel GMRES for the EDD strategy, the standard (sequential) description is given first. For the pre-processed linear equations, Eq. 11, the GMRES solution process is essentially a least-squares process, [7] which may be expressed as follows:

$$\min_{\mathbf{x}_s = \mathbf{x}_0 + \mathcal{K}(\mathbf{A}, \mathbf{r}_0)} ||\mathbf{b} - \mathbf{A}\mathbf{x}_s||_2 \tag{25}$$

and $\mathcal{K}_s(\mathbf{A}, \mathbf{r}_0) \equiv span\{\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \cdots, \mathbf{A}^{s-1}\mathbf{r}_0\}$ where $\mathcal{K}_s(\mathbf{A}, \mathbf{r}_0)$ is an s-dimensional Krylov subspace [7], and $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$ is the initial residual employing $\mathbf{x}_0$ as the initial approximation to the required solution. The GMRES solves Eq.25 by constructing an orthonormal basis $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_s\}$ for $\mathcal{K}_s(\mathbf{A}, \mathbf{r}_0)$ using the Arnoldi algorithm [7] and then finding the optimum solution based on this basis. In this paper a variant of the GMRES which is known as the flexible GMRES (FGMRES) [7] is employed. This variant permits the easy construction of different preconditioners at required stages in the iterative process of the GMRES solver. FGMRES differs from GMRES in that the solution updates are constructed using the preconditioned variable $\mathbf{z}_j$ instead of basis $\mathbf{v}_j$. The sequential implementation of the FGMRES is described in Algorithm 1.

Algorithm 1 *Sequential flexible GMRES with restart and preconditioner*

```
(1)  Start:  choose $\mathbf{x}_0$, $\check{m}$
        (dimension of the Krylov subspace),
        setup $(\check{m}+1)$-by-$\check{m}$ matrix $\tilde{H}$.
(2)  Arnoldi process:
(3)     $\mathbf{r}_0 = \mathbf{b} - \mathbf{A}\mathbf{x}_0$;  $\beta = ||\mathbf{r}_0||_2$;  $\mathbf{v}_0 = \mathbf{r}_0/\beta$;
(4)     FOR $j = 0, \cdots, \check{m}-1$ DO until convergence
(5)        $\mathbf{z}_j = \mathbf{C}\mathbf{v}_j$;
```


Ω3
Ω4
Ω1
Ω2
$\acute{K}1$ or $\hat{K}1$
$\acute{K}2$ or $\hat{K}2$
$\acute{K}3$ or $\hat{K}3$
$\acute{K}4$ or $\hat{K}4$
$\acute{v}1$ or $\hat{v}1$
$\acute{v}2$ or $\hat{v}2$
$\acute{v}3$ or $\hat{v}3$
$\acute{v}4$ or $\hat{v}4$

(a) finite element mesh partition (b) Associated matrix partitions

Figure 4: Non-overlapping finite element-based domain decomposition.

```
(6)             $\mathbf{v}_{j+1} = \mathbf{A}\mathbf{z}_j$
(7)             FOR $(i = 0, \cdots, j)$ $h_{i,j} = \langle \mathbf{v}_{j+1}, \mathbf{v}_i \rangle$;
(8)             FOR $(i = 0, \cdots, j)$ $\mathbf{w}_j = \mathbf{w}_j - h_{i,j}\mathbf{v}_i$;
(9)             $h_{j+1,j} = \|\mathbf{v}_{j+1}\|_2$; $\mathbf{v}_{j+1} = \mathbf{v}_{j+1}/h_{j+1,j}$;
(10)        ENDFOR
(11)        Define $\mathbf{Z}_j = [\mathbf{z}_i, \cdots, \mathbf{z}_j]$ and
            $\tilde{\mathbf{H}}_j = \{h_{i,k}\}$ $(0 \leq i \leq k+1, 0 \leq k \leq j)$;
(12)  Compute the approximate solution:
(13)        $\mathbf{x}_j = \mathbf{x}_0 + \mathbf{Z}_j\mathbf{y}_j$
            such that $\mathbf{y}_j = arg\min_{\mathbf{y}} \|\beta e_1 - \hat{H}_j\mathbf{y}\|_2$;
(14)  Restart:  IF (not convergent) $\mathbf{x}_0 = \mathbf{x}_j$ and
            GOTO (2);
```

From the above discussions it is clear that the polynomial preconditioner has a significant potential to many practical problems. However, for it to be readily employed for finite element applications involving large scale problems an efficient parallel implementation is required. In the next section, an efficient implementation of the polynomial preconditioning based GMRES for finite element-based partitioning is presented. This is followed by the parallel implementation for row-partitioning of the matrix that results due to the finite element node-based partitioning.

3 Finite element-based domain decomposition based GMRES

In this section, the distributed formulation of linear systems, Eq. 1, arising from non-overlapping finite element-based domain decomposition strategy is discussed. Typical steps involved in finite element-based domain decomposition computation are listed in Algorithm 2 for illustration. The time-consuming kernels in iterative methods require an efficient implementation which depends on three basic operations. These are vector operations namely, vector-update and inner-product, matrix-vector product, and the preconditioning solver phase. This is followed by norm-1 diagonal scaling technique in conjunction with the polynomial preconditioned FGMRES discussed next.

Algorithm 2 *Finite element-based domain decomposition method*

```
(1)  Discretize the domain;
(2)  Partition $\Omega$ into $\mathcal{P}$ non-overlapping
     sub-domains in terms of element;
(3)  Compute elemental stiffness matrix $\mathbf{K}_e$;
```

```
(4)  Compute sub-domain stiffness matrix
     $\mathbf{K}^{(s)}$ from $\mathbf{K}_e$;
(5)  Apply boundary condition over $\partial\Omega^{(s)}\setminus\Gamma$;
(6)  Solve the system of equations employing
     preconditioned GMRES solver;
```

3.1 Data structures and computational kernels

3.1.1 Data structures for Matrix/vector operations

The following two definitions are required for a further discussion.

Definition 1 *A global variable, vector* $\mathbf{u}$ *or matrix* $\mathbf{K}$ *in a local distributed format is denoted by* $\acute{\mathbf{u}}^{(s)}$ *and* $\acute{\mathbf{K}}^{(s)}$ *if it only contains the local data.*

Definition 2 *A global variable, vector* $\mathbf{u}$ *or a matrix* $\mathbf{K}$ *in a global distributed format is denoted as* $\hat{\mathbf{u}}^{(s)}$ *and* $\hat{\mathbf{K}}^{(s)}$ *if it contains the data that are same at the sub-domain interfaces.*

These definitions are explained as follows. Consider a 1-D finite element truss problem made of two elements as shown in Figure 5(a). It follows that a global vector $\mathbf{u}$ can be formulated as global distributed vectors $\hat{\mathbf{u}}^{(s)}$ $(s=1,2)$ or local distributed vector $\acute{\mathbf{u}}^{(s)}$ $(s=1,2)$. In addition, Figure 5(b) shows that the global distributed vectors store the full value of the global vector $\mathbf{u}$ with respect to the subdomain s, while the local distributed vector contains the data only attributed to the local elements in the sub-domain s. It should be noted that $\acute{\mathbf{u}}^{(s)} \neq \hat{\mathbf{u}}^{(s)}$.

Let $\mathbf{B}_s \in \Re^{N_s\times N}$ represent an unsigned boolean matrix which maps the global vector into the sub-domain s. It follows that

$$\hat{\mathbf{u}}^{(s)} \equiv \mathbf{B}_s\mathbf{u}; \quad \text{and} \quad \mathbf{u} \equiv \sum_{s=1}^{\mathcal{P}} \mathbf{B}_s^T \acute{\mathbf{u}}^{(s)} \tag{26}$$

A local distributed vector $\acute{\mathbf{u}}^{(s)}$ can be converted into a global distributed vector $\hat{\mathbf{u}}^{(s)}$ through

$$\hat{\mathbf{u}}^{(s)} \equiv \mathbf{B}_s \sum_{s=1}^{\mathcal{P}} \mathbf{B}_s^T \acute{\mathbf{u}}^{(s)} \tag{27}$$

This operation can be efficiently implemented by restricting the communication context within the neighboring sub-domains of s represented as

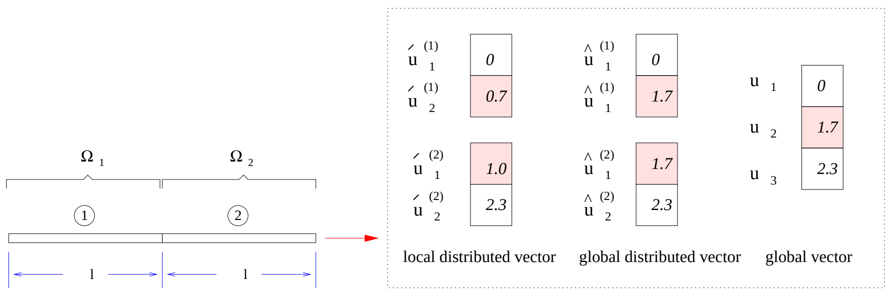
$$\hat{\mathbf{u}}^{(s)} = \bowtie \sum^{\partial\Omega_s} \acute{\mathbf{u}}^{(s)} \tag{28}$$

where $\partial\Omega_s$ is the interface of the sub-domain s. Here we refer to this operation as a nearest neighbor communication [6]. For details of the implementation the reader is referred to [6]. Analogous to vectors, a global stiffness matrix $\mathbf{K}$ also consists of a series of global distributed sub-matrices $\hat{\mathbf{K}}^{(s)}$ or a series of local distributed sub-matrices $\acute{\mathbf{K}}^{(s)}$. Again, referring to the 1-D truss problem (Figure 5(a)), the global stiffness matrix $\mathbf{K}$, the local distributed stiffness matrix, $\acute{\mathbf{K}}^{(s)}$ $(s=1,2)$, and $\hat{\mathbf{K}}^{(s)}$ $(s=1,2)$ which is the global distributed stiffness matrix are given by

$$\mathbf{K} = \frac{\mathcal{AE}}{l}\begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \tag{29}$$

$$\acute{\mathbf{K}}^{(1)} = \frac{\mathcal{AE}}{l}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \acute{\mathbf{K}}^{(2)} = \frac{\mathcal{AE}}{l}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{30}$$

$$\hat{\mathbf{K}}^{(1)} = \frac{\mathcal{AE}}{l}\begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}, \hat{\mathbf{K}}^{(2)} = \frac{\mathcal{AE}}{l}\begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}, \tag{31}$$



(a) 1-D FEM truss problem (b) Local and global distributed vector

Figure 5: Data-structure for EDD strategy.

where l is the element length, $\mathcal{A}$ is cross sectional area, and $\mathcal{E}$ is Young's modulus. It is clear that, in the general case,

$$\hat{\mathbf{K}}^{(s)} = \mathbf{B}_s^T \mathbf{K} \mathbf{B}_s; \quad \text{and} \quad \mathbf{K} = \sum_{s=1}^{\mathcal{P}} \mathbf{B}_s^T \acute{\mathbf{K}}^{(s)} \mathbf{B}_s \tag{32}$$

Here $\mathbf{K}$ is the global stiffness matrix for a domain Ω with homogeneous Neumann boundary conditions on the inner boundaries $\partial\Omega$.

3.1.2 Vector, matrix-vector and preconditioning solver operations

In the implementation of iterative methods for the solution of a linear system of equations, four basic computational kernels, namely, the vector update, inner-product, matrix-by-vector product and preconditioning are needed. This section discusses their efficient implementation for element-based domain decomposition.

Vector operations

Vector updates do not require any communication and the left-hand-side vector can be always determined by the local quantities, i.e., DAXPY ($\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y}$) is accomplished via $\mathbf{y}^{(s)} \leftarrow \alpha\mathbf{x}^{(s)} + \mathbf{y}^{(s)}, s = 1, 2, \cdots, \mathcal{P}$.

The inner product operation $\langle \mathbf{x}, \mathbf{y} \rangle$ where $\mathbf{x}, \mathbf{y} \in \Re^N$, is

$$\begin{aligned} \langle \mathbf{x}, \mathbf{y} \rangle &= \langle \textstyle\sum_{s=1}^{\mathcal{P}} \mathbf{B}_s \acute{x}^{(s)}, \sum_{s=1}^{\mathcal{P}} \mathbf{B}_s \acute{y}^{(s)} \rangle \\ &= \textstyle\sum_{s=1}^{\mathcal{P}} (\acute{\mathbf{y}}^{(s)})^T (\mathbf{B}_s^T \sum_{s=1}^{\mathcal{P}} \mathbf{B}_s \acute{\mathbf{x}}^{(s)}) \\ &= \textstyle\sum_{s=1}^{\mathcal{P}} \langle \acute{\mathbf{y}}^{(s)}, \hat{\mathbf{x}}^{(s)} \rangle \end{aligned} \tag{33}$$

Similarly, it can be shown that

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{s=1}^{\mathcal{P}} \langle \acute{\mathbf{x}}^{(s)}, \hat{\mathbf{y}}^{(s)} \rangle \tag{34}$$

It then follows from Eq. 33 and 34 that if both $\mathbf{x}$ and $\mathbf{y}$ are stored in a local distributed format, then $\gamma = \langle \mathbf{x}, \mathbf{y} \rangle$ can be computed via

$$\begin{cases} (1) & \hat{\mathbf{y}}^{(s)} = \bowtie \sum^{\partial\Omega_s} \acute{\mathbf{y}}^{(s)} \\ (2) & \gamma^{(s)} = \langle \acute{\mathbf{x}}^{(s)}, \hat{\mathbf{y}}^{(s)} \rangle \\ (3) & \gamma = \forall \sum \gamma^{(s)} \end{cases} \tag{35}$$

Matrix-by-vector operation

The matrix-vector operation $\mathbf{y} \leftarrow \mathbf{Kx}$, where $\mathbf{x}, \mathbf{y} \in \Re^N$ and $\mathbf{K} \in \Re^{N\times N}$, is the only stiffness-matrix-related subroutine in this implementation. Since

$$\begin{aligned} \mathbf{Kx} &= (\sum_{s=1}^{\mathcal{P}} \mathbf{B}_s^T \acute{\mathbf{K}}^{(s)} \mathbf{B}_s)(\sum_{s=1}^{\mathcal{P}} \mathbf{B}_s^T \acute{\mathbf{x}}^{(s)}) \\ &= \sum_{s=1}^{\mathcal{P}} \mathbf{B}_s^T \acute{\mathbf{K}}^{(s)} \hat{\mathbf{x}}^{(s)} \end{aligned} \tag{36}$$

then storing $\mathbf{y}$ in a local distributed format such that $\mathbf{y} \equiv \sum_{i=1}^{\mathcal{P}} \mathbf{B}_s^T \acute{\mathbf{y}}^{(s)}$, it follows that

$$\acute{y}^{(s)} = \acute{\mathbf{K}}^{(s)} \hat{\mathbf{x}}^{(s)} \tag{37}$$

Preconditioning
The application of the preconditioning is similar to that of a matrix-vector product. If the preconditioner $\mathbf{C}$, is available in a local distributed matrix format, $\mathbf{C}^{(s)}$, then $\mathbf{C} = (\sum_{s=1}^{\mathcal{P}} \mathbf{B}_s^T \mathbf{C}^{(s)} \mathbf{B}_s)$. It readily follows that, the preconditioning operation can be written as

$$\begin{aligned} \mathbf{w} = \mathbf{Cu} &= (\sum_{s=1}^{\mathcal{P}} \mathbf{B}_s^T \acute{\mathbf{C}}^{(s)} \mathbf{B}_s)(\sum_{s=1}^{\mathcal{P}} \mathbf{B}_s^T \acute{\mathbf{u}}^{(s)}) \\ &= \sum_{s=1}^{\mathcal{P}} \mathbf{B}_s^T (\acute{\mathbf{C}}^{(s)} \mathbf{B}_s \sum_{s=1}^{\mathcal{P}} \mathbf{B}_s^T \acute{\mathbf{u}}^{(s)}) \end{aligned} \tag{38}$$

This implies that

$$\mathbf{w}^{(s)} = \acute{\mathbf{C}}^{(s)} \mathbf{B}_s \sum_{s=1}^{\mathcal{P}} \mathbf{B}_s^T \acute{\mathbf{u}}^{(s)} = \acute{\mathbf{C}}^{(s)} \hat{\mathbf{u}}^{(s)} \tag{39}$$

Note that, it is assumed that a preconditioner is available in each processor which does not need nearest neighboring or global communication for its construction.

3.2 Parallel implementation

In this section, the parallel implementation of the diagonally scaled polynomial preconditioned flexible GMRES solver employing the previously discussed data structures and computational kernels is presented.

3.2.1 Norm-1 Diagonal-scaling

If the original problem is partitioned according to the finite element-based domain decomposition, then Eq. 2 in terms of the local distributed format yields

$$\left\{ \bigcup_{s=1}^{\mathcal{P}} \acute{\mathbf{K}}^{(s)} \right\} \left\{ \bigcup_{s=1}^{\mathcal{P}} \acute{\mathbf{u}}^{(s)} \right\} = \bigcup_{s=1}^{\mathcal{P}} \acute{\mathbf{f}}^{(s)} \tag{40}$$

The partition $\mathbf{K} = [\mathbf{k}_1, \mathbf{k}_2, \cdots, \mathbf{k}_N]^T$ described in the Theorem 1 in terms of local distributed format yields $\acute{\mathbf{K}}^{(s)} = [\mathbf{k}_1^{(s)}, \mathbf{k}_2^{(s)}, \cdots, \mathbf{k}_{N_s}^{(s)}]^T$. Let norm-1 diagonal scaling matrix $\mathbf{D}$ be formulated in the global distributed format, i.e., $\mathbf{D} = \bigcup_{s=1}^{\mathcal{P}} \hat{\mathbf{D}}^{(s)}$. Then

$$\hat{\mathbf{D}}^{(s)} = \text{diag}(1/\sqrt{\hat{d}_1^{(s)}}, \ldots, 1/\sqrt{\hat{d}_{N_s}^{(s)}}) \tag{41}$$

where

$$\hat{d}_i^{(s)} = \bowtie \sum^{\partial\Omega_s} \acute{d}_i^{(s)} \tag{42}$$

$$\acute{d}_i^{(s)} = \|\mathbf{k}_i^{(s)}\|_1 = \sum_{j=1}^{N_s} | k_{i,j}^{(s)} | \tag{43}$$

Eqs. 41 and 43 can be readily computed using the following algorithm.

Algorithm 3 *Construction of Diagonal Scaling*

```
(1)  FOR ($i = 1, 2, \cdots, N_s$) $\acute{\mathbf{d}}_i^{(s)} = \|\acute{\mathbf{k}}_i^{(s)}\|_1$;
(2)  assign $\acute{d}^{(s)} = [\acute{d}_1^{(s)}, \cdots, \acute{d}_{N_s}^{(s)}]^T$;
(3)  $\hat{d}^{(s)} = \bowtie \sum^{\partial\Omega_s} \acute{d}^{(s)}$;
(4)  FOR ($i = 1, 2, \cdots, N_s$) $\hat{d}_i^{(s)} = \frac{1}{\sqrt{\hat{d}_i^{(s)}}}$;
```

Employing Algorithm 4, the norm-1 diagonal scaling matrix $\mathbf{D}$ can transform Eq. 40 into a linear system:

$$\left\{\bigcup_s^p \acute{\mathbf{A}}^{(s)}\right\}\left\{\bigcup_s^p \acute{\mathbf{x}}^{(s)}\right\} = \bigcup_s^p \acute{\mathbf{b}}^{(s)}, \tag{44}$$

where $\acute{\mathbf{A}}^{(s)} = \hat{\mathbf{D}}^{(s)}\acute{\mathbf{K}}^{(s)}\hat{\mathbf{D}}^{(s)}$, $\acute{\mathbf{x}}^{(s)} = \hat{\mathbf{D}}^{(s)}\acute{\mathbf{u}}^{(s)}$ and $\acute{\mathbf{b}}^{(s)} = \hat{\mathbf{D}}^{(s)}\acute{\mathbf{f}}^{(s)}$. The polynomial preconditioned solvers based on norm-1 diagonal scaling is described by the algorithm below:

Algorithm 4 *Application of Diagonal Scaling*

```
(1)  $\acute{\mathbf{A}}^{(s)} = (\hat{\mathbf{D}}^{(s)})^T\acute{\mathbf{K}}^{(s)}\hat{\mathbf{D}}^{(s)}$;
(2)  $\acute{\mathbf{f}}^{(s)} = (\hat{\mathbf{D}}^{(s)})^T\acute{\mathbf{f}}^{(s)}$;
(3)  Construct preconditioning polynomial $P_m$
     with $\Theta = (0, 1)$;
(4)  Solve $(\bigcup_s^p \mathbf{A}^{(s)})(\bigcup_s^p \acute{\mathbf{x}}^{(s)}) = \bigcup_s^p \mathbf{b}^{(s)}$ using
     EDD-FGMRES with preconditioner $P_m(\mathbf{A})$;
(5)  $\acute{\mathbf{u}}^{(s)} = \hat{\mathbf{D}}^{(s)}\acute{\mathbf{x}}^{(s)}$;
```

Based on assumption that $\Theta = (0, 1)$, a parallel polynomial preconditioned FGMRES for Eq. 44 is formulated and described below.

3.2.2 FGMRES for element-based partitioning

Following Algorithm 1 and the computational kernels described above, the finite element-based domain decomposition flexible GMRES solver for the system of equations, Eq. 44, can be readily derived which is listed in Algorithm 5.

Algorithm 5 *Restarted, polynomial preconditioned (P_m) EDD-GMRES:*

```
(1)   PART1: Initialization
(2)       choose $\acute{\mathbf{x}}_0^{(s)}$,
          $\breve{m}$ (dimension of the Krylov subspace);
          convergent = FALSE;
(3)   PART2: Arnoldi process
          /* SECTION 2.1:  Formulate $\acute{\gamma}_0^{(s)}$ */
(4)       $\hat{\mathbf{w}}^{(s)} = \bowtie \sum^{\partial\Omega_s} \acute{\mathbf{x}}_0^{(s)}$;
(5)       $\acute{\mathbf{v}}_0^{(s)} = \acute{\mathbf{b}}^{(s)} - \acute{\mathbf{A}}^{(s)}\hat{\mathbf{w}}^{(s)}$;
(6)       $\hat{\mathbf{w}}^{(s)} = \bowtie \sum^{\partial\Omega_s} \acute{\mathbf{v}}_0^{(s)}$;
(7)       $\beta = (\forall \sum^{\Omega} \langle \hat{\mathbf{w}}^{(s)}, \acute{\mathbf{v}}_0^{(s)} \rangle)^{\frac{1}{2}}$;
(8)       $\acute{\mathbf{v}}_0^{(s)} = \acute{\mathbf{v}}_0^{(s)} / \beta$;
          /* SECTION 2.2:
          Classical Gram-Schmidt process */
(9)       FOR $j = 0, \cdots, \breve{m} - 1$ DO
(10)          /* preconditioning Algorithm 7 */
              $\acute{\mathbf{z}}^{(s)} \leftarrow P_m(\acute{\mathbf{A}}^{(s)})\acute{\mathbf{v}}^{(s)}$
(11)          $\hat{\mathbf{w}}^{(s)} = \bowtie \sum^{\partial\Omega_s} \acute{\mathbf{z}}_j^{(s)}$;
(12)          /* matrix-vector product */
              $\acute{\mathbf{v}}_{j+1}^{(s)} = \acute{\mathbf{A}}^{(s)}\hat{\mathbf{w}}^{(s)}$
(13)          $\hat{\mathbf{w}}^{(s)} = \bowtie \sum^{\partial\Omega_s} \acute{\mathbf{v}}_{j+1}^{(s)}$;
(14)          FOR ($i = 0, ..., j$) $h_{i,j} = \forall \sum^{\Omega} \langle \hat{\mathbf{w}}^{(s)}, \acute{\mathbf{v}}_i^{(s)} \rangle$;
(15)          FOR ($i = 0, ..., j$) $\acute{\mathbf{v}}_{j+1}^{(s)} = \acute{\mathbf{v}}_{j+1}^{(s)} - h_{i,j}\acute{\mathbf{v}}_i^{(s)}$;
```

```
(16)            $\hat{\mathbf{w}}^{(s)} = \bowtie \sum^{\partial\Omega_s} \acute{\mathbf{v}}^{(s)}_{j+1}$;
(17)            $h_{j+1,j} = (\forall\sum^{\Omega}\langle \acute{\mathbf{v}}^{(s)}_{j+1}, \hat{\mathbf{w}}^{(s)}\rangle)^{\frac{1}{2}}$;
                                  /* $h_{j+1,j} = \|\acute{\mathbf{v}}^{(s)}_{j+1}\|_2$ */
(18)            IF $h_{j+1,j} \leq \varepsilon$ THEN
(19)                convergent = TRUE; BREAK;
(20)            ENDIF
(21)            /* normalize $\acute{\mathbf{v}}^{(s)}_{j+1}$ */
                $\acute{\mathbf{v}}^{(s)}_{j+1} = \acute{\mathbf{v}}^{(s)}_{j+1}/h_{j+1,j}$;
(22)        ENDFOR
            /* SECTION 2.3:  Formulate Krylov
                subspace (spanned by $\{\mathbf{z}_j\}_{j=1}^{m}$)
                and Heisenberg matrix $\tilde{\mathbf{H}}_m$*/
(23)        Define $\mathbf{Z}^{(s)}_j = [\acute{\mathbf{z}}^{(s)}_0, \cdots, \acute{\mathbf{z}}^{(s)}_j]$ and
            $\tilde{\mathbf{H}}_j = \{h_{i,k}\}$  $(0 \leq i \leq k+1, 0 \leq k \leq j)$;
(24)    PART3: Compute the approximate solution:
(25)        $\acute{\mathbf{x}}^{(s)}_j = \acute{\mathbf{x}}^{(s)}_0 + \mathbf{Z}^{(s)}_j \mathbf{y}_j$ such that
            $\mathbf{y}_j = arg\min_{\mathbf{y}} \|\beta e_1 - \tilde{\mathbf{H}}_j \mathbf{y}\|_2$;
(26)    PART4: Restart:
(27)        IF (not convergent) THEN
(28)            $\acute{\mathbf{x}}^{(s)}_0 = \acute{\mathbf{x}}^{(s)}_j$; GOTO (3);
(29)        ENDIF
```

A typical Arnoldi iteration consists of $\breve{m}$ Gram-Schmidt loops where $\breve{m}$ is the dimension of the Krylov subspace. Therefore for $\breve{m}$ Gram-Schmidt computations, there are $O(\breve{m}^2)$ inner-products to form the Heisenberg matrix $\tilde{\mathbf{H}}_{\breve{m}}$ A naive implementation will result in $O(\breve{m}^2)$ collective communications. This overhead will increase as the number of processors increases as the collective communication overhead is $O(\log \mathcal{P})$. Therefore, an efficient implementation is required which is achieved by restricting this overhead to $O(\breve{m})$ collective communications in statement 14 of Algorithm 5.

In the Algorithm 5, for each Arnoldi iteration in the statements 9-24, there exist three nearest neighbor communication operations $\bowtie \sum^{\partial\Omega_s}$ namely, statements 13, 15 and 18. In order to enhance the parallel scalability of the Algorithm 5 further, these operations should be minimized. This can be achieved by introducing preconditioning vectors $\mathbf{z}$ in the global distributed format instead of the basis vectors in the global distributed format. This reduces the nearest neighbor communication operation to one instead of three. The details of this enhanced parallel GMRES solver is described in Algorithm 6.

Algorithm 6 *Enhanced EDD-FGMRES with restart and preconditioner P_m:*

```
(1)     PART1: Initialization
(2)         choose $\hat{\mathbf{x}}^{(s)}_0$,
            $m$ (dimension of the Krylov subspace);
            convergent = FALSE;
(3)     PART2: Arnoldi process
            /* SECTION 2.1:  Formulate $\acute{\mathbf{v}}^{(s)}_0$ */
(4)         $\acute{\mathbf{v}}^{(s)}_0 = \acute{\mathbf{b}}^{(s)} - \acute{\mathbf{A}}^{(s)}\hat{\mathbf{x}}^{(s)}_0$;
(5)         $\hat{\mathbf{v}}^{(s)} = \bowtie \sum^{\partial\Omega_s} \acute{\mathbf{v}}^{(s)}_0$;
(6)         $\beta = (\forall\sum^{\Omega}\langle \hat{\mathbf{v}}^{(s)}_0, \acute{\mathbf{v}}^{(s)}_0\rangle)^{\frac{1}{2}}$;
(7)         $\acute{\mathbf{v}}^{(s)}_0 = \acute{\mathbf{v}}^{(s)}_0/\beta$;  $\hat{\mathbf{v}}^{(s)}_0 = \hat{\mathbf{v}}^{(s)}_0/\beta$
            /* SECTION 2.2:
            classical Gram-Schmidt process */
(8)         FOR $j = 0, \cdots, \breve{m}-1$ DO until convergence
            /* polynomial preconditioning Algorithm 7 */
(9)             $\acute{\mathbf{z}}^{(s)}_j \leftarrow P_m(\acute{\mathbf{A}}^{(s)})\hat{\mathbf{v}}^{(s)}_j$;
(10)            $\hat{\mathbf{z}}^{(s)}_j = \bowtie \sum^{\partial\Omega_s} \acute{\mathbf{z}}^{(s)}_j$;
(11)            /* matrix-vector product */
                $\acute{\mathbf{v}}^{(s)}_{j+1} = \acute{\mathbf{A}}^{(s)}\hat{\mathbf{z}}^{(s)}_j$;
(12)            $\hat{\mathbf{v}}^{(s)}_{j+1} = \bowtie \sum^{\partial\Omega_s} \acute{\mathbf{v}}^{(s)}_{j+1}$;
```

```
(13)            FOR $(i=0,...,j)$ $h_{i,j} = \forall\sum^{\Omega}\langle\hat{\mathbf{v}}^{(s)}_{j+1}, \acute{\mathbf{v}}^{(s)}_{i}\rangle$;
(14)            FOR $(i=0,...,j)$ DO
(15)                $\acute{\mathbf{v}}^{(s)}_{j+1} = \acute{\mathbf{v}}^{(s)}_{j+1} - h_{i,j}\acute{\mathbf{v}}^{(s)}_{i}$;
(16)                $\hat{\mathbf{v}}^{(s)}_{j+1} = \hat{\mathbf{v}}^{(s)}_{j+1} - h_{i,j}\hat{\mathbf{v}}^{(s)}_{i}$;
(17)            ENDFOR
                /* Gram-Schmidt process */
(18)            $h_{j+1,j} = (\forall\sum^{\Omega}\langle\acute{\mathbf{v}}^{(s)}_{j+1}, \hat{\mathbf{v}}^{(s)}_{j+1}\rangle)^{\frac{1}{2}}$;
(19)            IF $h_{j+1,j} \leq \varepsilon$ THEN
(20)                convergent = TRUE; BREAK;
(21)            ENDIF
(22)            $\acute{\mathbf{v}}^{(s)}_{j+1} = \acute{\mathbf{v}}^{(s)}_{j+1}/h_{j+1,j}$; $\hat{\mathbf{v}}^{(s)}_{j+1} = \hat{\mathbf{v}}^{(s)}_{j+1}/h_{j+1,j}$;
(23)        ENDFOR
            /* SECTION 2.3:
                Formulate Krylov subspace
                (spanned by $\{\mathbf{z}_j\}^{j}_{k=1}$)
                and Heisenberg matrix $\tilde{\mathbf{H}}_j$ */
(24)        Define $\mathbf{Z}^{(s)}_{j} = [\hat{\mathbf{z}}^{(s)}_{0}, \cdots, \hat{\mathbf{z}}^{(s)}_{j}]$ and
            $\tilde{\mathbf{H}}_j = \{h_{i,k}\}$ $(0 \leq i \leq k+1, 0 \leq k \leq j)$;
(25)    PART3: Compute the approximate solution:
(26)        $\hat{\mathbf{x}}^{(s)}_{j} = \hat{\mathbf{x}}^{(s)}_{0} + \mathbf{Z}^{(s)}_{j}\mathbf{y}_j$
            such that $\mathbf{y}_j = arg\min_{\mathbf{y}}\|\beta e_1 - \tilde{\mathbf{H}}_j\mathbf{y}\|_2$;
(27)    PART4: Restart:
(28)        IF (not convergent) THEN
(29)            $\hat{\mathbf{x}}^{(s)}_{0} = \hat{\mathbf{x}}^{(s)}_{j}$; GOTO (3);
(30)        ENDIF
```

3.2.3 Polynomial Preconditioning

The necessary polynomial preconditioning operation, in the Algorithm 5,line 10, and Algorithm 6, line 10, the Neumann-series polynomial preconditioning is described below. The application of the generalized least-squares polynomial preconditioning (whose construction is discussed in [15] is formulated in a similar manner.

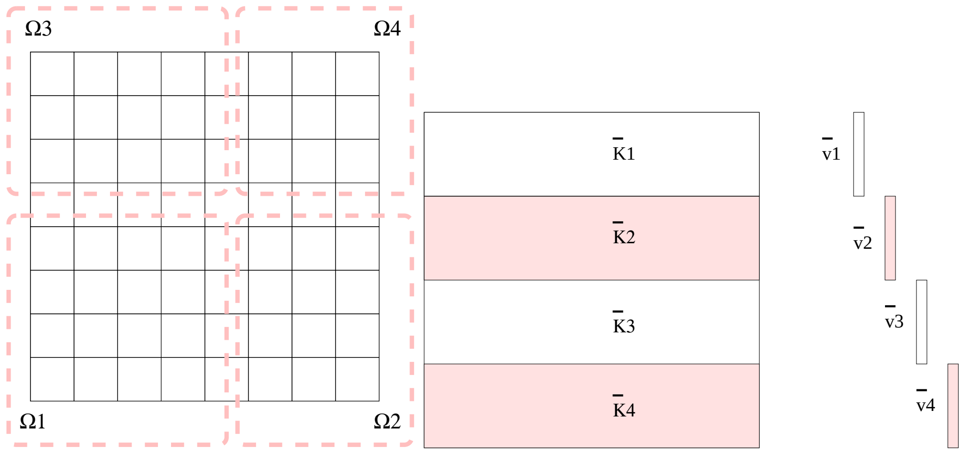
Algorithm 7 EDD Neumann Series Polynomial Preconditioning $\acute{\mathbf{z}}^{(s)} \leftarrow P_m(\acute{\mathbf{A}}^{(s)})\acute{\mathbf{v}}^{(s)}$. *m and* $\Theta = (0, \bar{h})$ *are input parameters.*

```
(1)  $\omega = \frac{1}{\bar{h}}$;
(2)  $\hat{\mathbf{w}}^{(s)} = \bowtie\sum^{\partial\Omega_s}\acute{\mathbf{v}}^{(s)}$;
(3)  $\acute{\mathbf{z}}^{(s)} = \acute{\mathbf{v}}^{(s)} - \omega\acute{\mathbf{A}}^{(s)}\hat{\mathbf{w}}^{(s)}$;
(4)  FOR $(i = 1, 2, \cdots, m)$ DO
(5)      $\acute{\mathbf{z}}^{(s)} = \acute{\mathbf{z}}^{(s)} + \acute{\mathbf{v}}^{(s)}$;
(6)      $\hat{\mathbf{w}}^{(s)} = \bowtie\sum^{\partial\Omega_s}\acute{\mathbf{z}}^{(s)}$;
(7)      $\acute{\mathbf{z}}^{(s)} = \acute{\mathbf{z}}^{(s)} - \omega\acute{\mathbf{A}}^{(s)}\hat{\mathbf{w}}^{(s)}$;
(8)  ENDFOR
(9)  $\acute{\mathbf{z}}^{(s)} = \omega(\acute{\mathbf{v}}^{(s)} + \acute{\mathbf{z}}^{(s)})$;
```

Alternative preconditioning methods such as ILU and SPAI may face drawbacks for finite element based domain decomposition methods. For example, the incomplete ILU preconditioner in the local distributed format computed via

$$(\mathbf{C}^{(s)})^{-1} \equiv \mathbf{L}^{(s)}\mathbf{U}^{(s)} = \acute{\mathbf{K}}^{(s)} \tag{45}$$

may occasionally suffer from singularity of $\acute{\mathbf{K}}^{(s)}$. In structural mechanics, if sub-domain s does not contain sufficient number of Dirichlet boundary conditions such that it "floats", then $\acute{\mathbf{K}}^{(s)}$ will be singular.



(a) Finite element-node based partitioning (b) Associated matrix row partition

Figure 6: Storage format of row-base domain decomposition linear systems.

4 Row-based Partition Strategy

For the finite element method, if a node-based partitioning is employed (see Fig. 6(a)), then the associated matrix partitioning is a block row partitioning (see Fig. 6(b)). Efficient parallel implementation of iterative solvers based on block row partitioning of the matrix are implemented in many popular libraries such as such as pARMS [19], PSPARSLIB [18], Aztec [20], etc. To compare these implementations with the finite element based domain decomposition based GMRES described previously, the row-based partitioning implementation is presented in this section.

4.1 Data structures and computational kernels

4.1.1 Data structures

Row-based domain partitioning provides a solution method for a general purpose linear system of equations. Using specific graph methods [21], the degrees of freedom (d.o.f) and associated equations (rows of the stiffness matrix) are partitioned and assigned to different processors. Referring to Figure 6, let the original domain be partitioned into $\mathcal{P}$ nodally disconnected sub-domains. Then

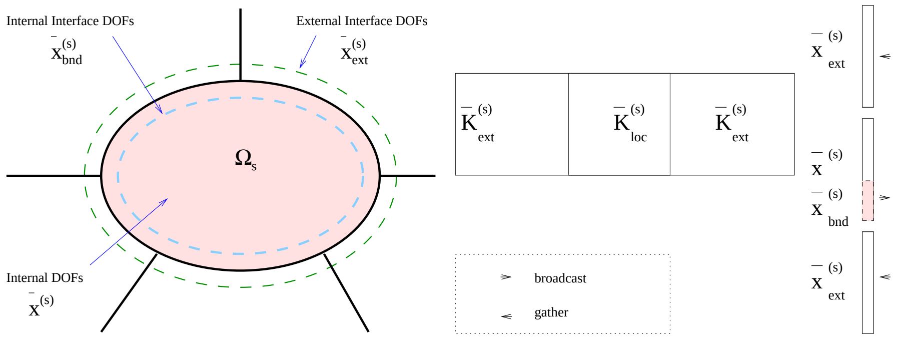
$$\overline{\mathbf{K}}^{(s)} = \mathbf{B}_s\mathbf{K}; \quad \text{and} \quad \overline{\mathbf{u}}^{(s)} = \mathbf{B}_s\mathbf{u} \tag{46}$$

It is clear from the above equations, that the matrix and vectors resulting due to finite element node-based partitioning is the row partitioning of the global matrix and vectors. Hence the matrix and vectors are always in the global distributed format.

4.1.2 Vector, matrix-vector, preconditioning operations

Similar to the element-based partitioning, the node-based partitioning also requires three basic operations, namely, vector, matrix-vector product, and preconditioning operations.
Vector operations
As in the element-based algorithm, the vector update operations for the RDD strategy do not require any interprocessor communications. The inner product

$$\langle \mathbf{x},\mathbf{y}\rangle = \sum_{s=1}^{\mathcal{P}} \langle \overline{\mathbf{x}}^{(s)}, \overline{\mathbf{y}}^{(s)}\rangle \tag{47}$$



(a) Adjacency of R-DD method (b) Matrix data structure of the sub-domain

Figure 7: Structures of the finite element row-based partitioning

is accomplished by a local inner product operation in each processor followed by a collective all reduce communication.

Matrix-vector operations
The Matrix-vector product operation of row-based partitioning is relatively more complex than that of element-based partitioning because the local d.o.f and the associated equations need to be de-coupled.
Figures 7(a) and 7(b) distinguishes three types of DOFs for row-based linear systems into interior DOFs ($\mathbf{x}^{(s)}_{loc}$), internal interface ($\mathbf{x}^{(s)}_{bnd}$) and external interface DOFs ($\mathbf{x}^{(s)}_{ext}$) [7, 19]. Further, as illustrated in Figure 7(a) and 7(b), the local equations are also de-coupled into two components, namely, the local matrices $\bar{\mathbf{K}}^{(s)}_{loc}$ and $\bar{\mathbf{K}}^{(s)}_{ext}$. The matrix-vector product operation $\mathbf{y} \longleftarrow \mathbf{Kx}$ can be accomplished by

$$\begin{cases} scatter\ \mathbf{x}^{(s)}_{bnd}\ to\ neighboring\ sub-domains; \\ gather\ \mathbf{x}^{(s)}_{ext}\ from\ neighboring\ sub-domains; \\ \bar{\mathbf{y}}^{(s)} = \mathbf{K}^{(s)}_{loc}\bar{\mathbf{x}}^{(s)}_{loc}; \\ \bar{\mathbf{y}}^{(s)} = \bar{\mathbf{y}}^{(s)} + \mathbf{K}^{(s)}_{ext}\bar{\mathbf{x}}_{ext}; \end{cases} \tag{48}$$

It should be noted that a re-ordering of local DOFs is required in order to achieve satisfactory parallel performance.
Precondition operations
In [19], additive Schwartz, Schur complement and ILU methods are employed as preconditioners for a row-based linear system . These methods are actually the extensions of the block Jacobi method [7] whose kernel is to solve the local system

$$\bar{\mathbf{K}}^{(s)}_{loc}\bar{\mathbf{z}}^{(s)} = \bar{\mathbf{v}}^{(s)}$$

In addition, an effective implementation of the GLS polynomial preconditioning is critically assessed in [15]. In the same manner as the EDD strategy, after employing the norm-1 diagonal scaling as a pre-process, the original linear system is transformed into

$$\left\{\bigcup_{s}^{\mathcal{P}}\bar{\mathbf{A}}^{(s)}\right\}\left\{\bigcup_{s}^{\mathcal{P}}\bar{\mathbf{x}}^{(s)}\right\} = \left\{\bigcup_{s}^{\mathcal{P}}\bar{\mathbf{b}}^{(s)}\right\} \tag{49}$$

Here, it should be remarked that the norm-1 diagonal scaling for the RDD strategy requires no inter-processor communication, while its matrix-vector product for the RDD strategy may incur more communication overhead than the EDD strategy.

4.1.3 FGMRES for row-based partitioning

Following Algorithm 1 and the computational kernels described above, the finite row-based domain decomposition flexible GMRES solver to solve the system of equation Eq. 49 can be readily derived which is listed in Algorithm 8.

Algorithm 8 *RDD-FGMRES with restart and polynomial preconditioner P_m:*

```
(1)  PART1: Initialization
(2)       choose $\bar{\mathbf{x}}_0^{(s)}$,
          $\breve{m}$ (dimension of the Krylov subspace),
               setup a $(\breve{m}+1)$-by-$\breve{m}$ matrix $\tilde{\mathbf{H}}$.
(3)  PART2: Arnoldi process
          /* SECTION 2.1:  Formulate $\bar{\mathbf{v}}_0^{(s)}$ */
(4)       $\bar{\mathbf{v}}_0^{(s)} = \bar{\mathbf{b}}^{(s)} - MatVec(\bar{\mathbf{A}}^{(s)}, \bar{\mathbf{x}}_0^{(s)})$; (Eq. 48)
(6)       $\beta = (\forall \sum^{\Omega} \langle \bar{\mathbf{v}}_0^{(s)}, \bar{\mathbf{v}}_0^{(s)} \rangle)^{\frac{1}{2}}$;
(7)       $\bar{\mathbf{v}}_0^{(s)} = \bar{\mathbf{v}}_0^{(s)}/\beta$;
          /* SECTION 2.2:
          Classical Gram-Schmidt process */
(8)       FOR $j = 0, \cdots, \breve{m}-1$ DO
(9)            /* polynomial preconditioning */
               $\bar{\mathbf{z}}_j^{(s)} = P_m(\bar{\mathbf{A}}^{(s)})\bar{\mathbf{v}}_j^{(s)}$;
(10)           /* matrix-vector product */
               $\bar{\mathbf{v}}_{j+1}^{(s)} = MatVec(\bar{\mathbf{A}}^{(s)}, \bar{\mathbf{z}}_j^{(s)})$
(11)           FOR $(i = 0, ..., j)$ $h_{i,j} = \forall \sum^{\Omega} \langle \bar{\mathbf{v}}_{j+1}^{(s)}, \bar{\mathbf{v}}_i^{(s)} \rangle$;
(12)           /* Gram-Schmidt process */
               FOR $(i = 0, ..., j)$ $\bar{\mathbf{v}}_{j+1}^{(s)} = \bar{\mathbf{v}}_{j+1}^{(s)} - h_{i,j}\bar{\mathbf{v}}_i^{(s)}$;
(13)           /* $h_{j+1,j} = \|\bar{\mathbf{v}}_{j+1}^{(s)}\|_2$ */
               $h_{j+1,j} = (\forall \sum^{\Omega} \langle \bar{\mathbf{v}}_{j+1}^{(s)}, \bar{\mathbf{v}}_{i+1}^{(s)} \rangle)^{\frac{1}{2}}$;
(14)           IF $h_{j+1,j} \leq \varepsilon$ THEN
                    convergent = TRUE; break;
               ENDIF
(15)           $\bar{\mathbf{v}}_{j+1}^{(s)} = \bar{\mathbf{v}}_{j+1}^{(s)}/h_{j+1,j}$; /* normalize$\bar{\mathbf{v}}_{j+1}^{(s)}$ */
(16)      ENDFOR
          /* SECTION 2.3:
          Formulate Krylov subspace
          (spanned by $\{z_j\}_{j=1}^{j}$)
               and Heisenberg matrix $\hat{H}_j$*/
(17)      Define $\mathbf{Z}_j^{(s)} = [\bar{\mathbf{z}}_0^{(s)}, \cdots, \bar{\mathbf{z}}_j^{(s)}]$ and
          $\tilde{\mathbf{H}}_m = \{h_{i,k}\}$ $(0 \leq i \leq k+1, 0 \leq k \leq j)$;
(18) PART3: Compute the approximate solution:
(19)      $\bar{\mathbf{x}}_j^{(s)} = \bar{\mathbf{x}}_0^{(s)} + \mathbf{Z}_j^{(s)} y_j$ such that
          $\mathbf{y}_j = arg\min_y \|\beta\mathbf{e}_1 - \tilde{\mathbf{H}}_j\mathbf{y}\|_2$;
(20) PART4: Restart:
          $\bar{\mathbf{x}}_0^{(s)} = \bar{\mathbf{x}}_j^{(s)}$;
(21)      IF (not convergent) THEN GOTO (3);
          /* $\bar{\mathbf{x}}_0^{(s)}$ stores the final solution */
```

5 Comparison of finite element and row-based domain decomposition

The computational complexity of each inner Arnoldi process of the GMRES solver for both element-based and row-based strategies are given in Table. 1. Asymptotically, both algorithms have the same parallel run time T_p; making both algorithms equally scalable on a wide range of parallel architectures for finite element analysis. However, the actual performance difference between the two algorithms depends upon the actual constants associated with complexity of communication overhead for the node and element grid partitioner on the same finite element grid.

In both algorithms, the main operations are `dAXPY` operations, vector inner-products computations and matrix-vector multiplication. The `dAXPY` operations do not involve any communication overhead. The communication time

Method	Neighbor comm.	Global comm.	mat-vec	Inner-product	vector-updates
Algorithm 5	m+3	$\breve{m}+1$	m+1	$O(\breve{m})$	$O(\breve{m}+m)$
Algorithm 6	m+1	$\breve{m}+1$	m+1	$O(\breve{m})$	$O(\breve{m}+m)$
Algorithm 8	m+1	$\breve{m}+1$	m+1	$O(\breve{m})$	$O(\breve{m}+m)$

Table 1: Communication and computation cost of inner Arnoldi process of GMRES solver ($\breve{m}$ is the size of Krylov subspace)

per iteration for a vector inner product depends only on the number of processors in use. This time is $O(\log \mathcal{P})$ on the hypercube and the HiPPI switch based architecture, and $O(\sqrt{\mathcal{P}})$ on a mesh based architecture. In the presence of a fast, hardware supported reduction operation we can assume that it is a small constant. Hence the only performance between element-based and row-based algorithms lies in matrix-vector product. For element-based matrix-vector multiplication, the communication only happens among the nearest neighboring processes; for row-based matrix-vector multiplication, more processes will get involved in communication.

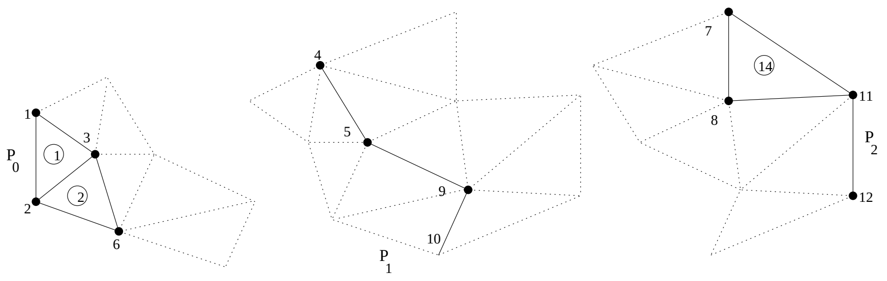


Figure 8: Element assignment to different processors for node based partition mesh.

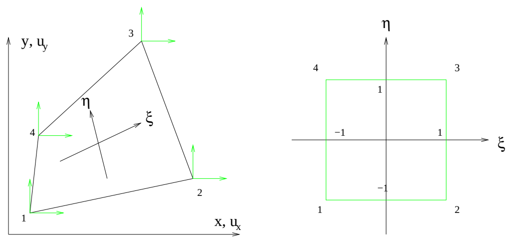
In finite element analysis for an unstructured finite element mesh the resulting coefficient matrix is sparse with an irregular sparsity pattern. Earlier work in sparse parallel matrix-vector multiplication [22] concluded that the efficiency of such algorithms depends on the data structure used and the sparsity pattern of the matrix **K**. Furthermore, a scalable parallel implementation of a sparse matrix-vector multiplication exists for a sparse matrix **K** provided that it is the adjacency matrix of a *planar graph* [1] $G(\mathbf{K})$ [21]. Since row-based GMRES algorithms require efficient parallel matrix-vector multiplication, both the search direction computation and the polynomial preconditioning computation which increases with the degree of polynomials for higher order elements such as a 4-noded quadrilateral and the 8-noded quadrilateral, etc., it makes $G(\mathbf{K})$ non-planar and thus deteriorates the scalability of parallel implementation. Furthermore, in finite element analysis, in contrast to the element-based algorithm the row-based algorithm has the following drawbacks. Referring to Figure 8, which shows elements assigned to each processor for the mesh, Figure 8(a) shows an example which is partitioned into three subdomains. Note that all the elements sharing the interface nodes must be assigned to the processors to avoid the assembly of interface contribution to the global matrix **K** which increases communication overhead due to gathering of entries of **K** from the neighboring processors resulting in duplicate assignment of the elements. This leads to two further drawbacks:

1. For large finite element meshes and especially for three-dimensional problems the computer storage requirements may increase drastically;

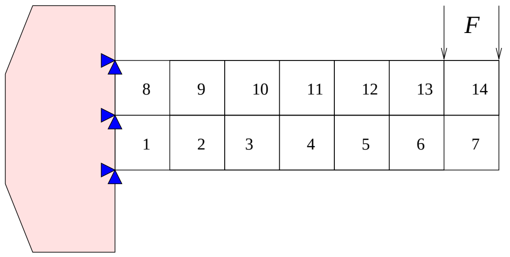
2. This will incur significant redundant floating-points operations for all the nodes assigned to the processors which are part of extra elements assigned and are not part of the original nodes in that partition.

It is clear from the above discussion that element domain decomposition based GMRES algorithms seem to be suitable for general parallel finite element analysis, especially for unsymmetric matrices.

[1] A graph is planar if and only if it can be drawn in a plane such that no edges cross each other. In finite element analysis, $G(\mathbf{K})$ is planar for a 3-noded triangular element.



(a) 4-node isoperimetrical quadrilateral element



(b) Cantilever plate

Figure 9: Problem description.

6 Numerical Results

In this section numerical results for two dimensional 2^{nd} order elasticity and elastodynamics problems employing the present parallel polynomial preconditioners are presented.

6.1 Problem description

Employing the finite element method for solving 2nd order elasticity, results in the discrete equation system

$$\mathbf{K}\mathbf{u} = \mathbf{f}, \tag{50}$$

where $\mathbf{K} \in \Re^{N\times N}$ is the global stiffness matrix, $\mathbf{u} \in \Re^{N}$ and $\mathbf{f} \in \Re^{N}$ are the displacement and externally applied load vectors respectively, and N is the number of degrees of freedom. Similarly, employing finite element formulation on the elastodynamics equations, results in following semi-discrete equation system

$$\mathbf{M}\frac{d^2\mathbf{u}}{dt^2} + \mathbf{K}\mathbf{u} = \mathbf{f};\ \mathbf{u}(0) = \mathbf{u}_0;\ \dot{\mathbf{u}} = \dot{\mathbf{u}}_0, \tag{51}$$

where $\mathbf{M} \in \Re^{N\times N}$ is a positive definite mass matrix. Observe that Equation 51 is a 2^{nd} order ordinary differential equation. Once time marching techniques such as the family of generalized integration operators [6] are employed for time discretization, it results in a system of linear equations of the form

$$\overline{\mathbf{M}}\mathbf{u}_{n+1} = [\alpha\mathbf{M} + \beta\mathbf{K}]\,\mathbf{u}_{n+1} = \hat{\mathbf{f}}_{n+1}, \tag{52}$$

where $\overline{\mathbf{M}}$ is the effective stiffness matrix, α and β are free parameters associated with specific time integration method, and $n+1$ is the current time step. For illustration, the Eqs. 50 – 52 are solved for a two-dimensional cantilever problem described in Fig. 9(a) employing four-node quadrilateral finite elements. The parallel flexible GMRES algorithms described previously are implemented using C and MPI. The finite element meshes of increasing problem size are shown in Table 2 where `nXele`$\times$`nYele` describes the mesh dimension in the unit of element, `nNode` is the total number of nodes in the mesh and `nEqn` is the degrees of freedom. Since all systems undergo diagonal scaling prior to computation of the solution, Θ can be simply defined as $\Theta = (\varepsilon, 1)$ where ε is the machine precision. The value of $\breve{m}$ (refer to Algorithm 1), the dimension of the Krylov subspace is chosen to be 25 and convergence is achieved when $\frac{\|\mathbf{r}_i\|_2}{\|\mathbf{r}_0\|_2} \leq tol$, where $\mathbf{r}_i$ is the i-th residual vector $(\mathbf{b} - \mathbf{A}\mathbf{x}_i)$. The value of $tol = 10^{-6}$ is chosen in the numerical experiments.

6.2 Results

The numerical experiments focus on:

Mesh	nXele$\times$nYele	nNode	nEqn
Mesh1	7×1	16	28
Mesh2	40×8	369	656
Mesh3	40×20	861	1,640
Mesh4	50×50	2,601	5,100
Mesh5	60×60	3,721	7,320
Mesh6	70×70	5,041	9,940
Mesh7	80×80	6,561	12,960
Mesh8	90×90	8,281	16,380
Mesh9	100×100	10,201	20,200
Mesh10	200×100	20,301	40,400

Table 2: Finite element mesh of increasing number of elements and nodes for a cantilever problem which is employed for the parallel performance studies for both EDD- and NDD-GMRES.

1. Comparison of polynomial and ILU(0) (ILU without any fill-in) [7] preconditioners;
2. Convergence improvement and the degree of polynomial preconditioners;
3. Parallel speedup of polynomial preconditioned EDD-FGMRES.

It should be remarked that, Θ, the spectrum estimation of the diagonally scaled linear system (11) is always regarded to be $(0,1)$ in the following experiments. However, as illustrated in Figure 10, making $\Theta = (0,1)$ does not necessarily lead to optimal convergence performance.

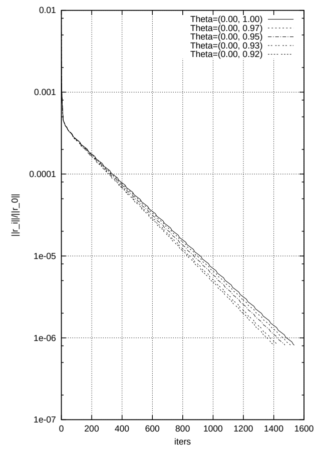


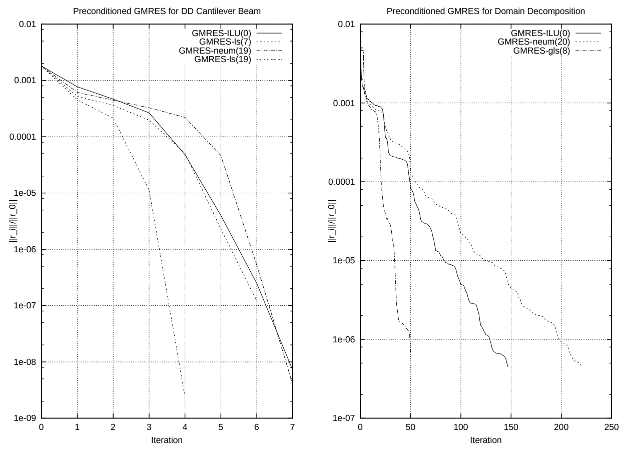
Figure 10: Convergence of EDD-GMRES-gls(10) versus the estimation about $\sigma(A)$ (Θ).

Polynomial Preconditioner vs. ILU(0) ILU preconditioner is considered to be one of the most efficient (from the point of view of convergence iteration) sequential preconditioning method [7]. While Figures 11 and 12 illustrate that, in the context of single processor,

$$GLS(7) \succ ILU(0) \succ Neum(20),$$

holds for both static and dynamic problems of `Mesh1` and `Mesh2`. Here "$\succ$" indicates "converge faster than", and the degree of polynomials are given bracket. As a conclusion, the performance of the GLS and Neumann series polynomial preconditioners is completely comparable to the ILU(0) preconditioner on a single processor.

Here it should be remarked that two small-scale systems `Mesh1` and `Mesh2` are chosen to study the convergence of solvers because ILU(0) does not work for a large-scale problem in element-based domain decomposition setting. Compared to polynomial preconditioners, the ILU requires more storage.



(a) 7×1 (b) 40×8

Figure 11: ILU(0) versus polynomial preconditioner for static analysis about cantilever beam with pulling load problem.

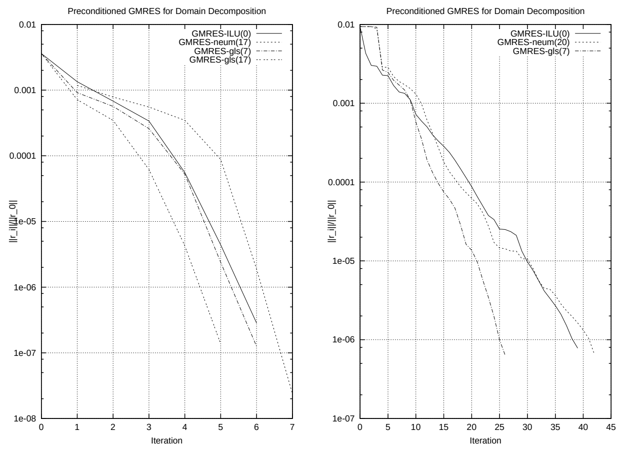
Degree of Polynomial and The Resulting Convergence Performance From both the Figures 13 and 14, it is observed that for both the static and dynamic problems of `Mesh1` and `Mesh2`,

$$GLS(20) \succ GLS(10) \succ GLS(7) \succ GLS(3) \succ GLS(1),$$

which implies that a higher degree GLS polynomial preconditioning may bring about more reduction of the iteration number for convergence. However, this is not be true for some relatively large problems. For example, as illustrated in Table 3, GLS(7) performs better than GLS(8), GLS(9) and GLS(10) for the static problem of **Mesh7**.

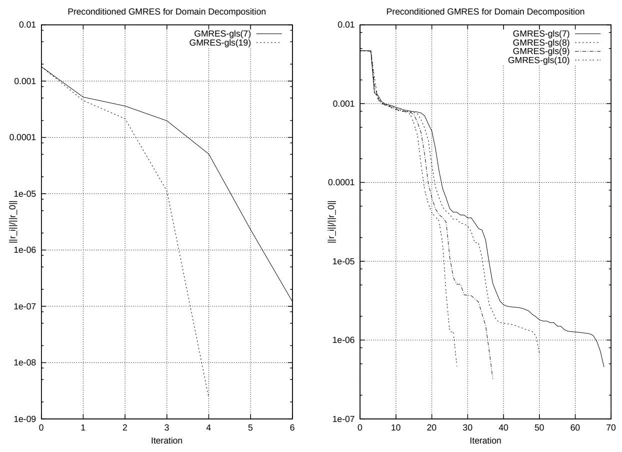
Parallel Speedup The results presented in Figure 17 and Table 3 show that the polynomial preconditioned EDD-FGMRES achieves an acceptable speedup when multiple processors are used on both the IBM-SP2 and the SGI-ORIGIN machine. Four important observations can be made. First, the parallel speedup of the polynomial preconditioned FGMRES is related to the degree of the preconditioning polynomial. Trivially, the higher the degree of the preconditioning polynomial, the more dominant the matrix-vector product will become in the whole problem. Figure 17 (a) shows that for the same kind of problem, EDD-FGMRES-GLS(m) has better parallel speedup than EDD-FGMRES-GLS(n) if $m > n$. In contrast, Figure 17 (b) shows that the value of m does not influence the parallel speedup of the RDD-FGMRES-GLS(m) significantly. Secondly, it is observed via Figures 17 (c) and (d) that, with the increase in the dimension of the problem, the parallel performance achieved will approach a higher level of linear speedup. This phenomenon is also illustrated in Table 3. Thirdly, as shown in Table 3, even though the GLS(10) has better convergence performance than the GLS(7), the EDD-FGMRES-GLS(10) has a significant time-cost than the GLS(7) because in each iteration the GLS(10) performs three additional matrix-vector product operations than GLS(7). Therefore, a trade-off between convergence performance and CPU time should be made.

From Figure 17(e) it is clear that our implementation is also highly portable. However, the parallel speedup depends on the HPC system in question. Figure 17(e) shows that the SGI-ORIGIN has a better parallel performance than the IBM-SP2, which can be attributed to the shared memory architecture of the SGI-ORIGIN machine as against the distributed memory architecture of the IBM-SP2. In the case that only a small number of processors participate



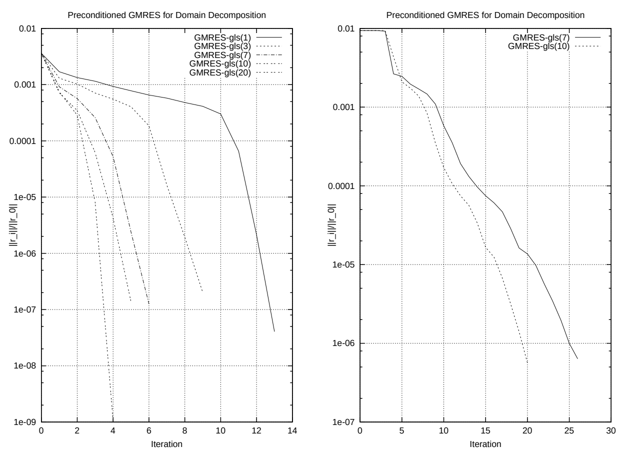
(a) 7×1 (b) 40×8

Figure 12: ILU(0) versus polynomial preconditioner for dynamic analysis about cantilever plate with pulling load problem.



(a) 7×1 (b) 40×8

Figure 13: Convergence versus increasing degree of preconditioning polynomial for static analysis of cantilever plate problem.



(a) 7×1 (b) 40×8

Figure 14: Convergence versus increasing degree of preconditioning polynomial for dynamic analysis about cantilever plate problem.

in the computation, the IBM-SP2 has a higher communication overhead between the computing nodes than the SGI-ORIGIN.

7 Conclusions

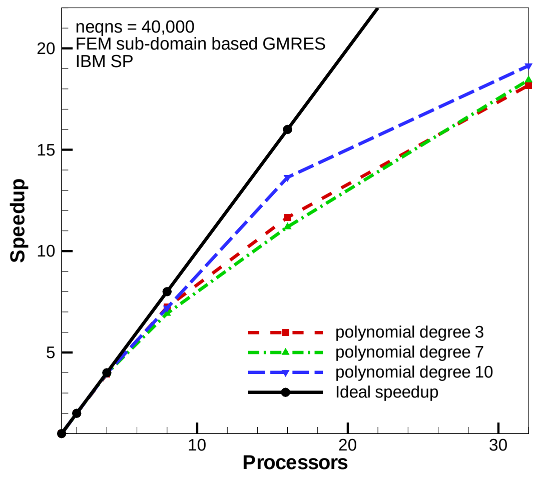
In this paper an efficient parallel finite element-based domain decomposition GMRES solver with polynomial preconditioning is presented.

- For single processor computations, the convergence performance (in terms of the number of iterations) of the polynomial preconditioned methods are comparable with the ILU(0) preconditioned ones. Further, for finite element based domain decomposition computations, polynomial preconditioner is a preferable choice as it is cheaper to construct than the ILU(0) preconditioner.
- The finite element based domain decomposition solver in conjunction with the polynomial preconditioning (such as the Neumann series, GLS, etc.) circumvents the following operations:
  - the assembly process for associated coefficient matrix and preconditioner;
  - reordering of degree of freedom;
  - redundant computations associated with the interface elements;
  - numerical problems associated with local ILU(k) preconditioner.

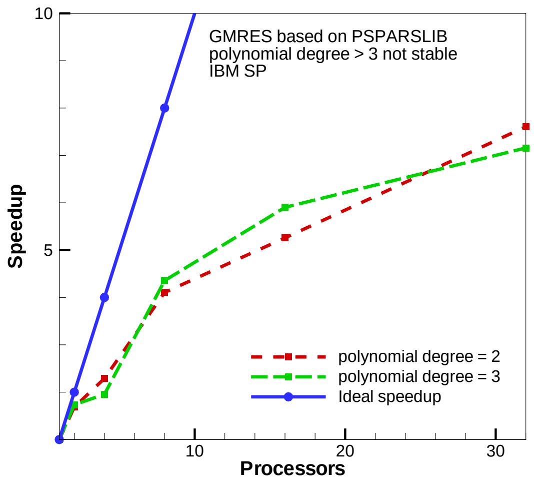
  This is unlike in the case of standard row-oriented partitioning strategies. As a consequence, a dramatic reduction in parallel overhead both in terms of computations and communications is achieved. The parallel performance results for illustrative large-scale static/dynamic problems on the IBM SP2 and the SGI Origin were presented.

Acknowledgements

The authors are very pleased to acknowledge support in part by Battelle/U. S. Army Research Office (ARO) Research Triangle Park, North Carolina, under grant number DAAH04-96-C-0086, and by the Army High Performance Computing Research Center (AHPCRC) under the auspices of the Department of the Army, Army Research Laboratory (ARL) under contact number DAAD19-01-2-004. The content does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. Special thanks are due to the CIS Directorate at the U. S. Army Research Laboratory (ARL), Aberdeen Proving Ground, Maryland. Other related support in form of computer grants from the Minnesota Supercomputer Institute (MSI), Minneapolis, Minnesota is also gratefully acknowledged.



(a) EDD with increasing *m*



(b) RDD with increasing *m*

Figure 15: Parallel speedup results employing polynomial preconditioned FGMRES with increasing degree of polynomials.

References

[1] K. J. Bathe. *Finite Element Procedures*. Englewood Cliffs, 1995.

[2] R. Kanapady and K. K. Tamma. A-scalability and an integrated computational technology and framework for non-linear structural dynamics. Part 1: Theoretical developments and parallel formulations . *International Journal for Numerical Methods in Engineering*, 58(15):2295–2323, 2003.

[3] R. Kanapady and K. K. Tamma. *A*-Scalability of an Integrated Computational Technology and Framework for Non-linear Structural Dynamics - Part I Theoretical Developments and Parallel Formulations. *Int. J. Numer. Methods Engrg*, 2003. (in press).

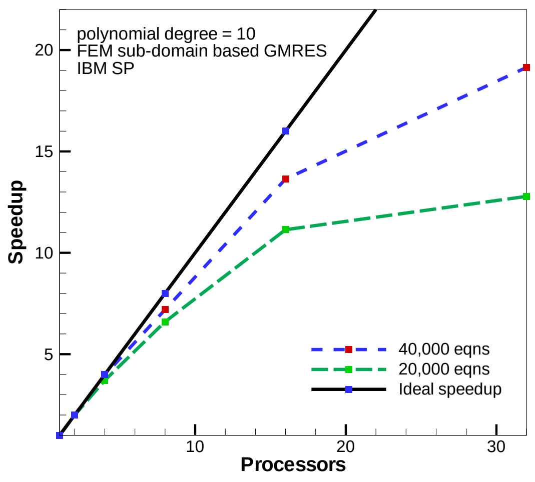
[4] Farhat C. and Roux F.-X. A Method of Finite ELement Tearing and Interconnecting and Its Parallel Solution Algorithm. *Int. J. Numer. Mech. Eng.*, pages 1205–1227, 1991.

[5] Yousef Saad. Iterative Solution of Indefinite Symmetric Linear Systems by Methods Using Orthogonal Polynomial over Two Disjoint Interval. *SIAM J. Numer. Anal.*, 20:784–811, 1983.

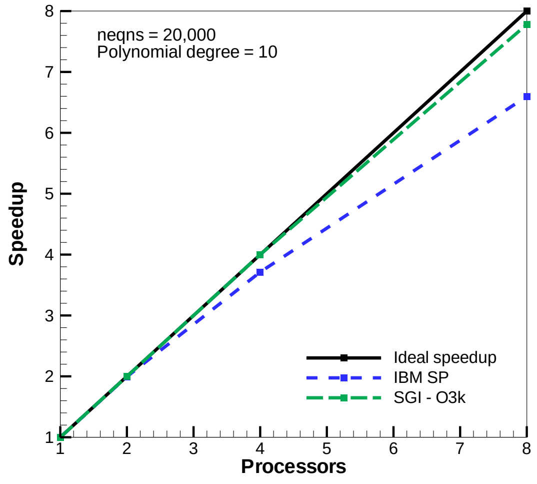
[6] R. Kanapady, K. K. Tamma, and A. Mark. Highly Scalable Parallel Computational Models for Large-scale RTM Process Modeling Simulations, PART 2: Parallel Formulation Theory and Implementation. *Numerical Heat Transfer Part B: Fundamentals*, 36(3):287–308, 1999.

Mesh	$\mathcal{P}$	Degree of Preconditioning Polynomial											
		7			8			9			10		
		Its.	Time	*S*	Its.	Time	*S*	Its.	Time	*S*	Its.	Time	*S*
1	1	68	1.2192	1.00	51	1.0253	1.00	37	0.8193	1.00	27	0.6633	1.00
	2	68	0.6904	1.77	51	0.5823	1.76	37	0.4609	1.78	28	0.3872	1.71
	4	69	0.4237	2.88	53	0.3710	2.76	38	0.2835	2.89	29	0.2421	2.74
	8	69	0.3352	3.64	53	0.2998	3.42	38	0.2398	3.42	30	0.2082	3.19
2	1	36	1.202	1.00	32	1.175	1.00	29	1.188	1.00	26	1.179	1.00
	2	36	0.611	1.97	32	0.678	1.73	29	0.653	1.82	26	0.647	1.82
	4	36	0.378	3.18	32	0.372	3.16	29	0.369	3.22	26	0.367	3.21
	8	36	0.241	4.99	32	0.239	4.92	29	0.233	5.10	26	0.217	5.43
3	1	98	9.966	1.00	80	9.150	1.00	71	9.043	1.00	57	7.924	1.00
	2	98	5.291	1.88	80	4.860	1.88	71	4.764	1.90	57	4.172	1.90
	4	98	2.840	3.51	80	2.591	3.53	71	2.544	3.55	57	2.250	3.52
	8	98	1.603	6.22	81	1.543	5.93	71	1.451	6.23	58	1.290	6.14
4	1	179	26.13	1.00	132	21.93	1.00	87	15.80	1.00	79	15.87	1.00
	2	179	13.75	1.90	132	11.29	1.94	87	8.26	1.91	79	8.23	1.93
	4	179	7.74	3.38	133	6.17	3.55	87	4.43	3.57	79	4.40	3.61
	8	176	4.22	6.19	133	3.39	6.47	88	2.60	6.08	79	2.46	6.45
5	1	181	46.39	1.00	179	51.32	1.00	148	47.11	1.00	113	39.69	1.00
	2	182	23.90	1.94	179	26.64	1.93	149	24.72	1.91	113	20.42	1.80
	4	181	12.59	3.68	179	14.47	3.55	149	12.71	3.71	114	10.69	3.43
	8	182	7.02	6.61	179	7.46	6.88	150	6.86	6.87	117	5.67	6.47
6	1	160	44.65	1.00	164	51.50	1.00	189	65.58	1.00	168	67.38	1.00
	2	160	21.59	2.07	164	25.81	2.00	189	33.22	1.97	169	33.43	2.02
	4	161	11.46	3.90	164	13.00	3.96	189	16.59	3.95	169	16.33	4.13
	8	161	6.10	7.32	164	7.13	7.22	189	8.70	7.54	169	8.68	7.76
7	1	241	82.24	1.00	159	60.34	1.00	159	67.02	1.00	186	90.02	1.00
	2	241	41.33	1.99	159	30.57	1.97	159	34.23	1.96	186	43.65	2.06
	4	241	21.27	3.87	159	16.04	3.76	159	18.15	3.69	186	22.60	3.98
	8	241	11.83	6.95	159	8.86	6.81	158	9.78	6.85	186	12.77	7.05

Table 3: Performance results, iterations, total CPU time (sec) and speedup (S) for FGMRES-GLS(m) for the static problem on SGI-Origin.



(a) For increasing problem size



(b) IBM SP and SGI Origin

Figure 16: Parallel speedup results employing polynomial preconditioned FGMRES.

[7] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing, 1996.

[8] Petter E Bjorstad Barry F Smith and William D Gropp. *Domain Decomposition*. Cambridge University Press, 2004.

[9] S. F. Ashby. *Polynomial preconditioning for conjugate gradient method*. PhD thesis, Univ. of Illinois at Urbane-Champaig, 1987.

[10] S. F. Ashby. Minimax Polynomial Preconditioning for Hermitian Linear System. *SIAM J. Matrix Anal. Appl*, 1991.

[11] Bernd Fischer and Gene H. Golub. On Generating Polynomials Which are Orthogonal over Several Interval. *Mathematics of Computation*, pages 711–730, 1991.

[12] Bernd Fischer. Polynomial Based Iteration Methods for Symmetric Linear Systems. In *Advances in Numerical Mathematics*. Wiley-Teubner Series, 1996.

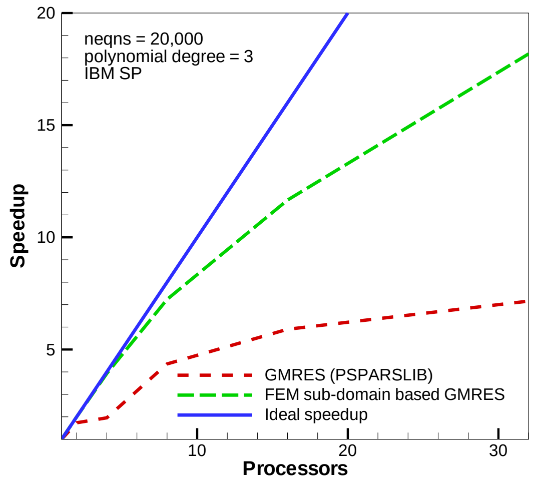
[13] Walter Gautschi. On Generating Orthogonal Polynomials. *SIAM J. Sci. Stat. Comput.*, 3(3), 1982.

[14] Hong Jiang. On the orthogonality of residual polynomials of minimax polynomial preconditioning. *Numerische Mathematik*, 1994.

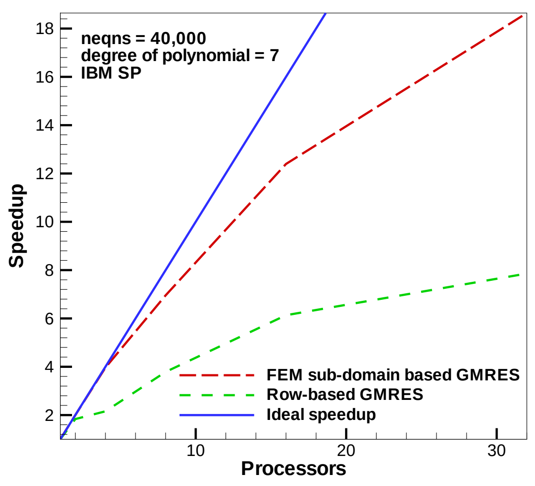
[15] Y. Liang, J. Weston, and M. Szularz. *Adaptive Generalized Least-squares Polynomial Preconditioner for Symmetric Indefinite Linear Equations*. *Parallel Computing*, 28(2):323–341, 2002.

[16] Y. Liang, J. Weston, and M. Szularz. Stability of Polynomial Preconditioning. In A. Handlovicova, M.Komornikova, K.Mikula, and D.Sevcovic, editors, *ALGORITMY 2000*, pages 264–273, 2000.

[17] Y. Saad. *pARMS: parallel Algebraic Recursive Multilevel Solvers*. http://www-users.cs.umn.edu/˜saad/software/pARMS/, version 1.0 edition.



(a) EDD and RDD(m=3)



(b) EDD and RDD(m=7)

Figure 17: Comparision of parallel speedup results of EDD and RDD employing polynomial preconditioned FGMRES.

[18] Y. Saad. *PSPARSLIBS: A Portable Library of Parallel Sparse Iterative Solvers*. http://www-users.cs.umn.edu/˜saad/research.html, 1999.

[19] Z. Li, Y. Saad, and M. Sosonkina. *pARMS : a parallel version of the algebraic recursive multilevel solver*. Technical Report UMSI-2001-100, Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN, 2001.

[20] J. N. Shadid and R. S. Tuminaro. Sparse iterative algorithm software for large-scale MIMD machines: an initial discussion and implementation. *Concurrency: Practice and Experience*, 4(6):481–497, 1992.

[21] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to parallel computing/ Design and analysis of Algorithms*. The Benjamin/Cummings Publishing Company, Inc., 1994.

[22] A. Gupta, V. Kumar, and A. Samesh. Performance and Scalability of Preconditioned Conjugate Gradient Methods on Parallel Computers. *IEEE Transactions on Parallel and Distributed Systems*, 1995.